NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTILEVEL DATA ASSOCIATION FOR THE VESSEL TRAFFIC SERVICES SYSTEM AND THE JOINT MARITIME COMMAND INFORMATION SYSTEM

by

Ian Neil Glenn

December, 1995

Thesis Advisor: Murali Tummala

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	**2. REPORT DATE** December 1995	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE MULTILEVEL DATA ASSOCIATION FOR THE VESSEL TRAFFIC SERVICES SYSTEM AND THE JOINT MARITIME COMMAND INFORMATION SYSTEM(U)	**5. FUNDING NUMBERS**
6. AUTHOR(S) Glenn, Ian Neil	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES) USCG Electronics Engineering Center, Wildwood, New Jersey	**10. SPONSORING/ MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

This thesis develops an algorithm to fuse redundant observations due to multiple sensor coverage of a vessel. Fuzzy membership functions are used as a measure of correlation, and a fuzzy associative system determines which observations represent the same vessel. The result is a computationally efficient algorithm. The output of the system is a unique set of vessels identified by unique platform identifiers. Results of tests based on computer simulation of overlapping radar coverage show that the fusion algorithm correctly correlates and fuses the sensor observations. That the VTS system is a subset of the Joint Maritime Command Information System (JMCIS) and ultimately the Global Command and Control Software (GCCS) system makes this algorithm pertinent not only to the US Coast Guard, but also to the US Navy, DOD and other agencies such as the Canadian Navy that use this software.

14. SUBJECT TERMS Vessel Traffic Services System, Data Fusion, Fuzzy Logic, Membership Function, Joint Maritime Command Information System, Global Command and Control System, Data Association			**15. NUMBER OF PAGES** 131
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

MULTILEVEL DATA ASSOCIATION FOR THE VESSEL TRAFFIC SERVICES SYSTEM AND THE JOINT MARITIME COMMAND INFORMATION SYSTEM

Ian Neil Glenn
Major, Canadian Armed Forces
B.Eng., Royal Military College of Canada, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1995

Author: [redacted]
Ian N. Glenn

Approved by: [redacted]
Murali Tummala, Thesis Advisor

[redacted]
Roberto Cristi, Second Reader

[redacted]
Herschel H. Loomis, Jr., Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

This thesis develops an algorithm to fuse redundant observations due to multiple sensor coverage of a vessel. Fuzzy membership functions are used as a measure of correlation, and a fuzzy associative system determines which observations represent the same vessel. The result is a computationally efficient algorithm. The output of the system is a unique set of vessels identified by unique platform identifiers. Results of tests based on computer simulation of overlapping radar coverage show that the fusion algorithm correctly correlates and fuses the sensor observations. That the VTS system is a subset of the Joint Maritime Command Information System (JMCIS) and ultimately the Global Command and Control Software (GCCS) system makes this algorithm pertinent not only to the US Coast Guard, but also to the US Navy, DOD and other agencies such as the Canadian Navy that use this software.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

The United States Coast Guard uses the US Navy's Joint Maritime Command Information System (JMCIS) software as the core software in their Vessel Traffic Services (VTS) systems. This software allows numerous sensors of various types, primarily radar, to make reports to the central supervisory and controlling site, the Vessel Traffic Center (VTC). At the VTC, the sensor information is plotted as tracks on the displays of the operators who are tasked with monitoring vessel traffic and providing advisories to vessels in transit or anchoring in key waterways. The current VTS software lacks a mechanism to correlate duplicate sensor tracks to reduce the amount of information presented to each operator. This thesis addresses this problem.

A. GOAL OF THESIS

This thesis presents an algorithm that performs central level fusion on data from various sensor sources providing vessel tracks for display and archival purposes. The algorithm is a refinement of a previously proposed algorithm to fuse the outputs of sensors, such as the Telephonics Remote Site Processor (RSP), providing overlapping coverage. The algorithm has been generalized to accept and fuse an arbitrary number of tracks from any available sensor that can provide any of the following feature information: latitude, longitude, course, speed, and size (approximately length times beam). The data collected are fused to create a single unified track table for display to the VTS operators and for maintenance of an historical record. The fusion process consists of several levels in order to achieve an integrated data set. Also, separate data conversion mechanisms are required to prepare the data for fusion. Figure 1.1 provides an overview of data flow within the Vessel Traffic Services system.

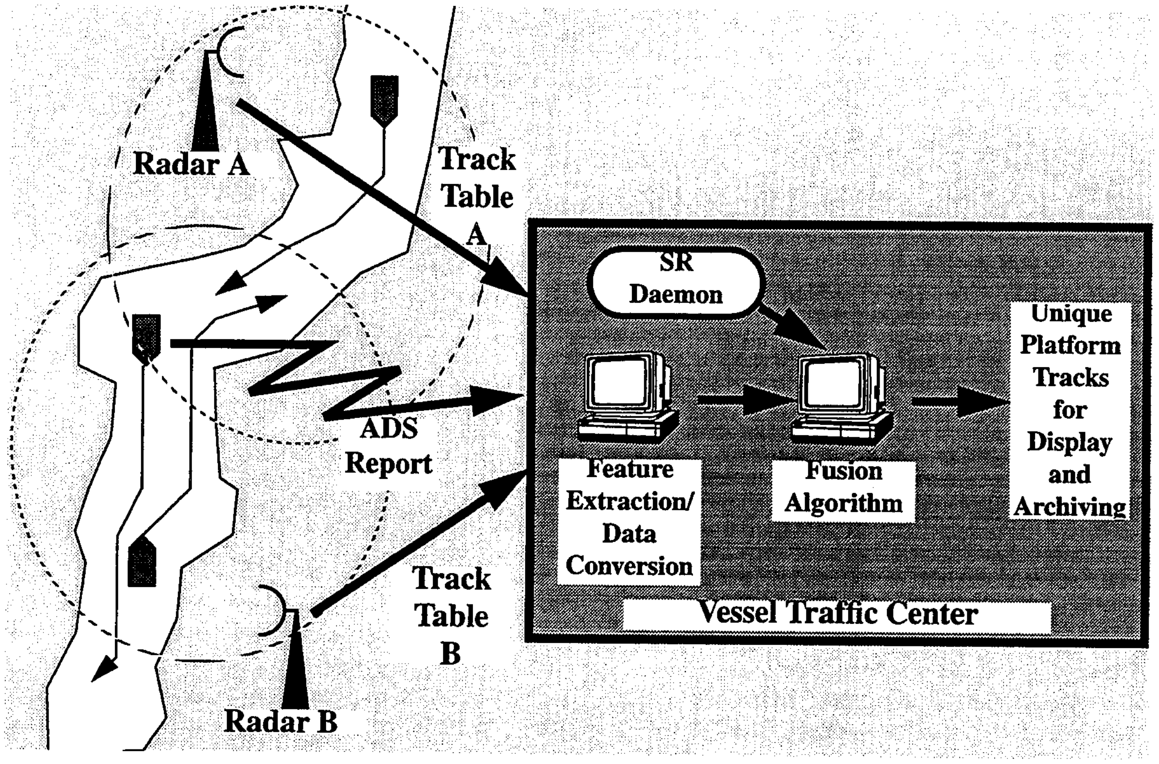


Figure 1.1. System Overview Diagram

B. THESIS OUTLINE

Figure 1.2 shows an outline of the proposed algorithm. Each section of the algorithm will be expanded upon in the following chapters. Sources of valid sensor tracks will be dealt with in Chapter II. The preprocessing necessary to prepare the data for the algorithm is discussed in Chapter III. The actual algorithm along with a description of the fuzzy association part of the algorithm is presented in Chapter IV. The simulation generated to test the performance of the algorithm along with the results is detailed in Chapter V. Conclusions are offered in Chapter VI. The Appendices list the fusion code and the simulation code used to test the algorithm.

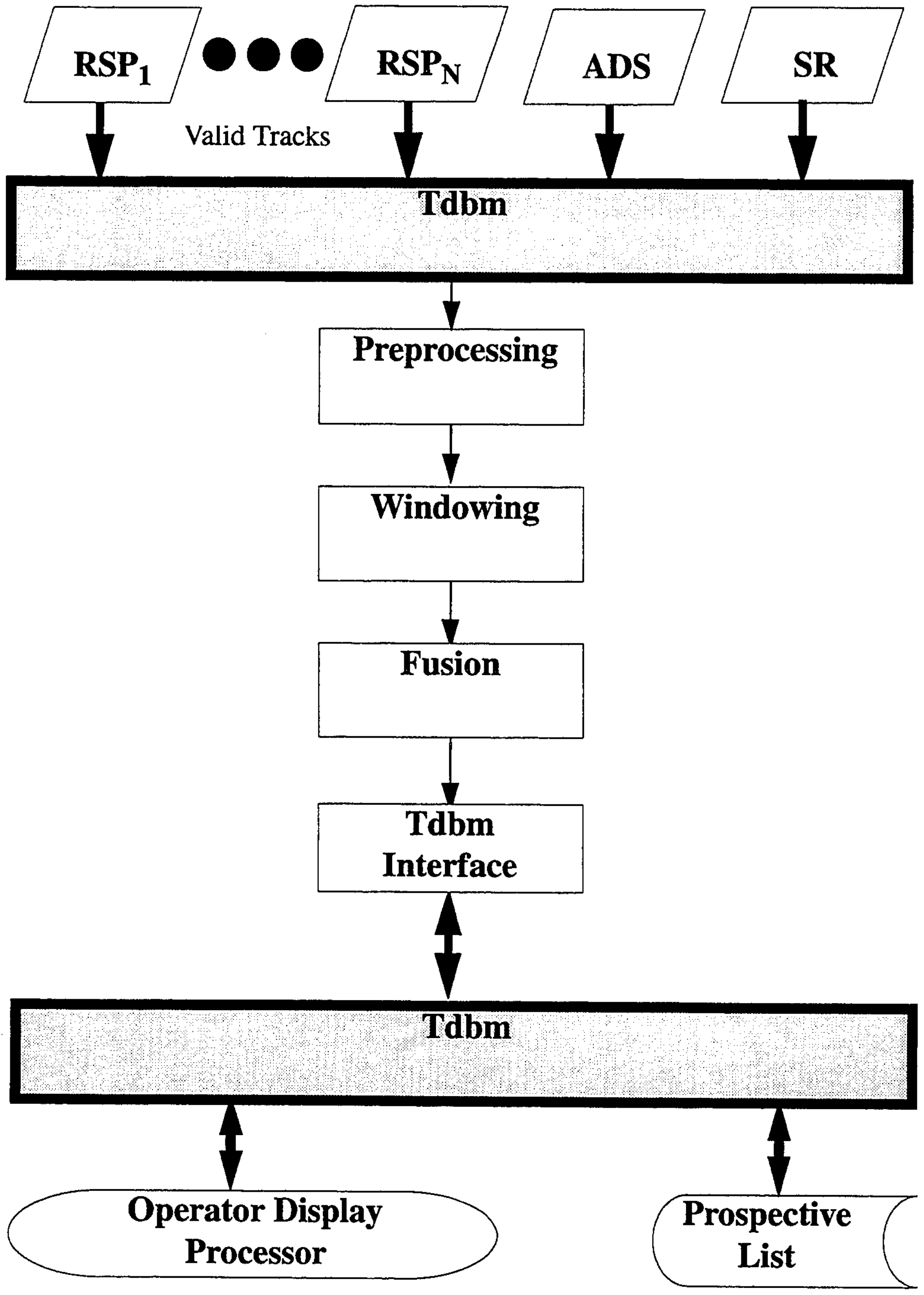


Figure 1.2. Overview of Fusion Algorithm

II. TRACK ACQUISITION

This chapter presents the details of how tracks are acquired by the VTS system. As the VTS system is a subset of the Joint Maritime Command Information System (JMCIS), the JMCIS software system is presented in detail with specific references to the particular setup of the VTS system — especially as it applies to the New York Harbor. The key element of the system is the Unified Build (UB) Software Development Environment (SDE) Track Database Manager (Tdbm) Service. [Ref. 6] The pertinent details of the Tdbm will be presented along with a discussion of the various sensor inputs that provide tracks to the Tdbm.

To appreciate the necessity of a data fusion algorithm, an understanding of the quantity and type of data that the various sensors are reporting continuously to the system is required. These sensors provide coverage of the Area of Responsibility (AOR) in electronic form, allowing automation of numerous supervisory and advisory tasks. In particular, hazardous situations can be automatically detected and the VTS operators alerted through the VTS Alarm Toolbox [Ref. 2].

The primary sources of vessel positions are the tracks generated by the Telephonics remote site processors (RSPs) running Kalman filters at each remote site [Ref. 8]. A new innovation under development is the Automatic Data Sensors (ADS) [Ref. 5] where vessels report either the Global Positioning System (GPS) or the Differential GPS (DGPS) information to the system over radio link. Estimated Positions (EPs) of vessels transiting through the region based on VTC established Standard Routes (SR) can also be made available to the system. These are referred to as SR tracks and are described fully later in this chapter.

As mentioned, the VTS software is essentially JMCIS with all correlation functions less Link Correlation turned off as indicated in Figure 2.1. Currently, the VTS operates as follows:

- Tracks are generated and reported to the central site, the Vessel Traffic Center.
- At the central site, tracks are fed through the link correlator where each track is considered a link.

Presently because there exists no automated way to perform link to link correlations, each link is associated on a one-on-one basis with a platform track. Platform tracks form the basis of what operators see on their displays and what is archived for historical purposes. The latter is performed by an archiving daemon that copies out the platform track contents of the Tdbm periodically to an archival database. Link tracks are not recorded anywhere in the system except in the Tdbm where they exist only until replaced by a more recent track reported by the same site and track number of a RSP.

Figure 2.1 illustrates the modules available in the JMCIS on which VTS is based. As previously stated, VTS uses only the link correlator and turns off all other functions to ensure that the one-to-one correlation between link track and platform track is not broken by the correlator. While this is required to ensure that the ship classifications do not disrupt the association of reports to platform tracks, it is limiting in that currently in VTS no many-to-one or multiple link tracks to one platform track associations can be made. The aim of the proposed algorithm is to perform those associations such that the association process is as transparent as possible to the operators.

Figure 2.2 shows in detail the inner workings of the JMCIS. The fusion algorithm would operate as part of the Correlator and interact directly with the Tdbm. The Track Database Manager provides the central clearinghouse for all sensor generated tracks reported to the central processing site. Tracks arrive and get recorded in the system. Each track is fed through the correlator to promote it to an existing platform track or cause the

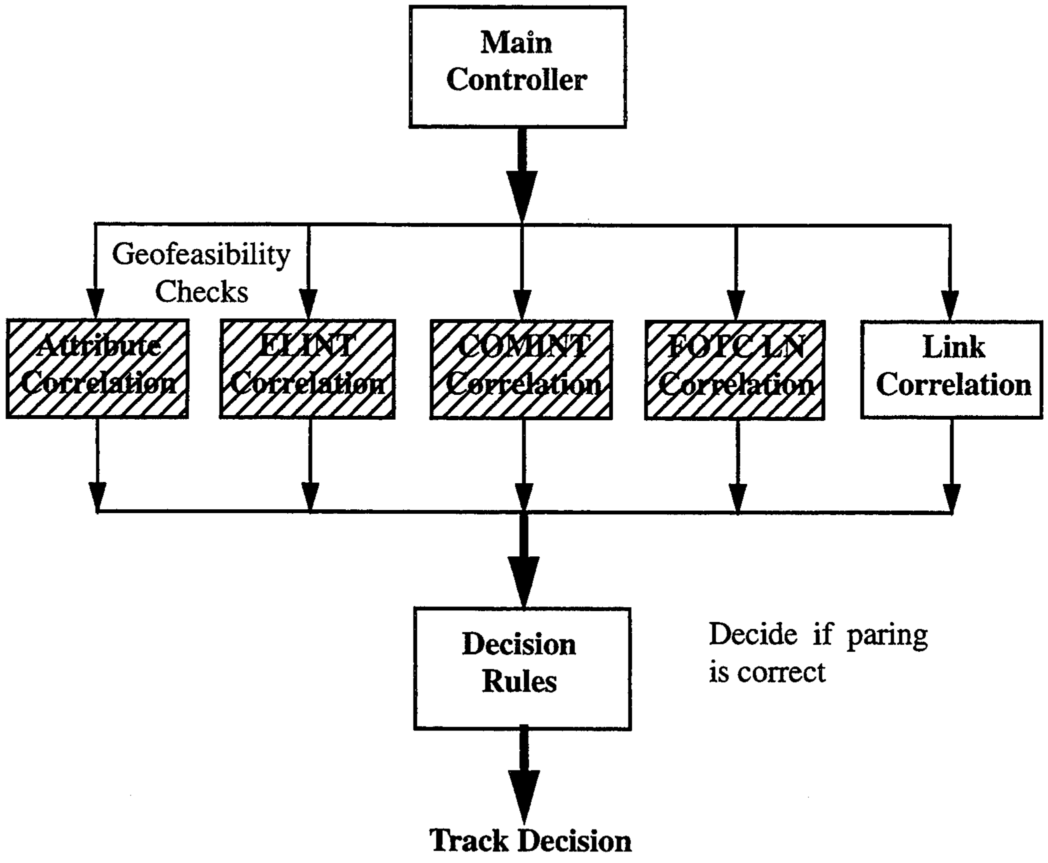


Figure 2.1. JMCIS Flow. From Ref. [7]

generation of a new platform track. In the current implementation, all linked tracks get promoted to platform tracks which are both archived and displayed. The proposed algorithm periodically examines the link tracks in the Tdbm. After processing, the output is a unique set of platform tracks with one-to-one promotions where only one sensor reports a vessel and many-to-one promotions where multiple sensors report the same vessel. As a result, only the actual vessels present are presented for display and archiving, and the best estimates of position, course, speed and size are used for those determinations.

The details of how the various sensor tracks, radar, ADS and SR, arrive in the Tdbm for processing are presented in the following

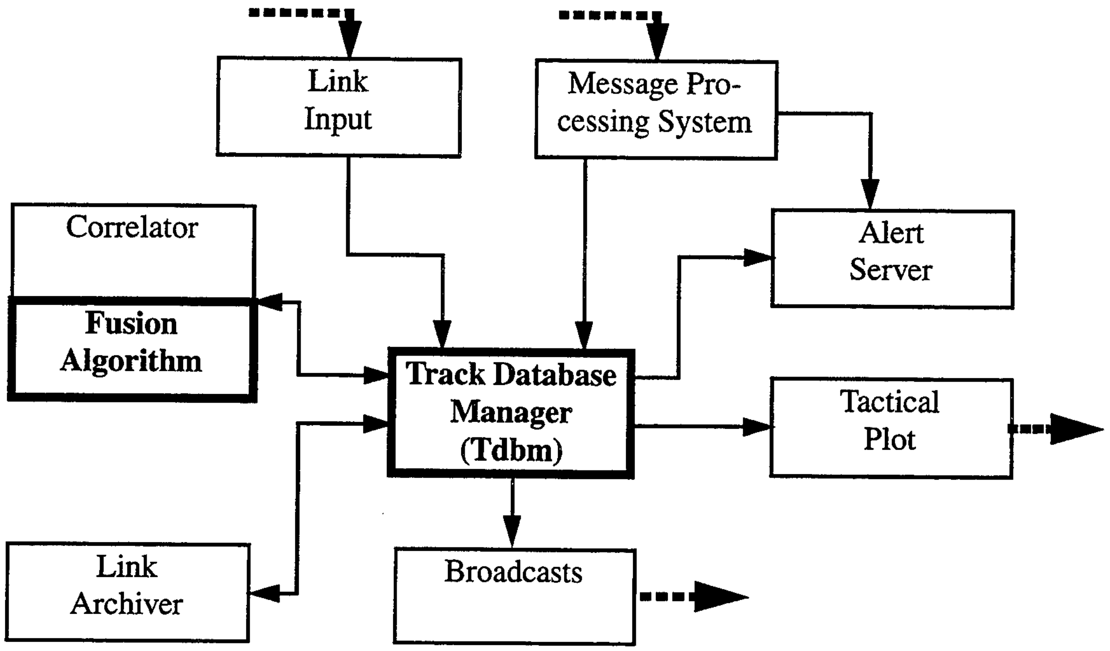


Figure 2.2. JMCIS Software Architecture

A. RADAR TRACKS

Radar tracks are generated by the Telephonics Radar Processor operating at each remote radar site in the VTS. The Radar Processor system is composed of two major processing elements. First is the Marine Target Extractor (MTE-2000) which performs the dual, independent functions of target extraction/tracking and scan conversion/image compression. Video imagery from the scan conversion/image compression portion of the MTE-2000 is not used in this algorithm, so that aspect of this device will not be discussed further. The second is the Maintenance Display and Control Unit (MDU) which provides a local control and display device. The limited raw data obtained for this thesis was recorded by connecting a Hewlett Packard Network Analyzer to the output of the MDU. [Ref. 8]

The MTE-2000 operates as follows. Radar inputs are accepted from the radar subsystem and routed to both the target extractor/tracker and scan converter/image compressor. The target extractor/tracker processes this data in the optimal three step process of hit detection, plot extraction and target tracking to maximize system sensitivity while minimizing the false alarm rate. Radar video is first sampled, passed through Sensitivity Time Control (STC) and Fast Time Constant (FTC) filters, and subjected to land blanking (to eliminate unwanted returns) and then a constant false alarm rate (CFAR) detection process. Video returns exceeding the CFAR detection threshold are classified as hits, and passed to the plot extractor along with the other target attributes. A distribution free (DF) quantizer is used for the CFAR processing to provide a constant false alarm rate independent of noise/clutter statistics. A clutter processor (CP) is used with the DF quantizer to dynamically adjust the detection parameters in preselected geographic area. [Ref. 8]

The plot extractor subsequently integrates hits over the antenna beamwidth using a sliding window intergrator and passes the resultant target plots to the target tracker. The target tracker provides target smoothing and tracking using an adaptive Kalman filter. The tracker provides both manual and automatic target acquisition. The details of the Target Detection Unit (TDU) and the Target Tracking Unit (TTU) will be expanded in the following paragraphs. Once calculated, track data, including target speed, heading and smoothed position, are output in Synchronous Data Link Control (SDLC) format to both the VTC and the MDU. [Ref. 8]

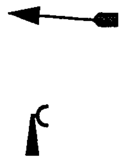
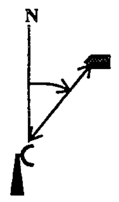
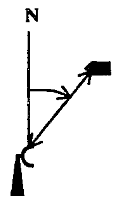
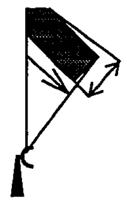
The Telephonics Radar Processor is designed to work with the Raytheon, Furuno and AIL radars in use in the VTS. Plot detection is accomplished by the TDU. It uses a sliding window detection algorithm to integrate hit data across the antenna beamwidth using leading edge, trailing edge and confidence count criteria to extract target plots from those hits thereby achieving CFAR detection. Leading edge (LE) detection is defined as the number of radar returns (hits) required in a specified window size to determine that a target has entered the antenna beam. Trailing edge (TE) detection is defined as the number of hits remaining in the window to establish that the antenna beam has gone past the target. Confidence count (CC) is the minimum number of hits required in an antenna beamwidth for a target to be declared a plot. This fusion algorithm assumes that these parameters have been optimized for detection while minimizing the false alarm rate. Default parameters for each of the radars used in the VTS to establish a false alarm rate of approximately 10^{-5} at the output of the TDU is shown in Table 2.1. [Ref. 8]

Radar	Mode	PRF	RPM	Ant BW (°)	Hits per BW	Window Size	LE Count	TE Count	CC
Raytheon	1	3600	20	0.45	14	13	5	3	6
	2	1800	20	0.45	7	8	4	2	4
	3	900	20	0.45	3	8	3	2	3
Furuno	1,2	2100	24	0.95	14	13	5	3	6
	3	1200	24	0.95	8	8	4	2	4
	4	600	24	0.95	4	8	3	2	3
AIL	1,2	2500	20	0.33	7	8	4	2	4

Table 2.1: TDU Detection Default Parameters. From Ref. [8].

In the next step, the Target Tracking Unit (TTU) uses a range-azimuth (rho-theta) Kalman filter algorithm to track targets once they have been plotted. Targets must progress though a three-step process (pairing, developing and maturing) before being output from the system. First a plot is paired or correlated with another plot on two successive antenna scans. These target pairs then develop, where there must be a range and azimuth correlation on M of the N next scans. Finally, successful targets are advanced to mature targets. Mature targets are updated and output every antenna scan and will continue to be tracked until they are either manually dropped by the operator or outside the radar coverage area. The track table data reported over the command/status channel to the VTC is current to one revolution of the antenna. [Ref. 8] A number of parameters may be configured to optimize the MTE-2000 for each radar and site. This thesis assumes valid tracks are being provided to the VTC Tdbm.

The format of reporting tracks to the Tdbm is shown in Table 2.2 based on information in [Ref. 8].

Site No	**Track No**	**Time of Track**	**Crse (°)**	**Speed (kts)**	**Pred Rge (nm)**	**Pred Az (°)**	**Rdr Rge (nm)**	**Rdr Az (°)**	**Extent Rge (nm)**	**Extent Az (°)**	**Track Quality**	**A Q**	**L T**	**C T**
(a)	**(b)**	**(c)**	**(d)**	**(e)**	**(f)**	**(g)**	**(h)**	**(i)**	**(j)**	**(k)**	**(l)**	**(m)**	**(n)**	**(o)**
1	**1**	**1512**	**263°**	**7**	**2.3**	**35°**	**2.3**	36°	**.5**	10°	**7**			**3**
2	**4**	**1512**	**90°**	**10**	**2.1**	127°	**2.0**	126°	**.1**	2°	**9**			
												**Acquisition**	**Lost**	**Coast**

Notes:

- **Track numbers** (b) are allocated to 1000, then recycled.
- **Time of Track** (c) is determined by GPS.
- **Track Quality** (l) is determined as follows: Start at 4 when target matures. Increment by one when hit occurs. Decrement by one when miss occurs.
- **AQ** (m): Acquisition Mode. A - Automatic; M - Manual.
- **LT** (n): Lost track. Set after the maximum number of coasts is exceeded. Indicates that this target is being dropped. The track number is now free to be reassigned. Can be used in fusion algorithm to clear track.
- **CT** (o): Coast track. When set, it indicates that there was no radar hit on that particular scan. After a predetermined number of misses, the LT bit is set.

Table 2.2: Telephonics Target Track Outputs. After Ref. [8].

B. AUTOMATED DEPENDENT SURVEILLANCE (ADS)

The Automated Dependent Surveillance (ADS) segment to VTS provides the capability to accept Global Positioning System (GPS) and Differential GPS (DGPS) track reports and generate and update JMCIS (VTS) tracks based on these reports [Ref. 5]. The motivation behind ADS is that a vessel's position, course and identification can be more easily provided to the Vessel Traffic Center by the vessel itself than solely by the VTC's own sensors. With ADS, the vessel sends its information to the VTC over a data link such as a satellite or digital selective calling (DSC). The National Marine Electronics Association (NMEA) 0183 Standard is used as the basis of the "Voiceless VTS" data packet used to report the ADS tracks. The NMEA 0183 defines a data reporting structure in the form of data "sentences". The specific structures to be used by ADS are the extended format NMEA messages given in Table 2.3. [Ref. 5]

From messages described in Table 2.3, the Vessel Position Report provides all of the features required by the fusion algorithm less size. The exact size of the reporting vessels can be determined through an Oracle database lookup of the vessel or fusion could be conducted with the size feature excluded.

The current plan in New York harbor involves data transmission using DSC on channel 70 at 9600 Baud. The anticipated update rate to the Tdbm is once every three seconds [Ref. 5] for each track. The system is designed to reject any reports that are not geofeasible. Phase I implementation of ADS, which integrates ADS with existing data structures, requires that the data in Table 2.4 be available for recording in the Tdbm.

NMEA Message		Description				
Data Field 1	**Data Field 2**	**Data Field 3**	**Data Field 4**	**Data Field 5**	**Data Field 6**	**Data Field 7**
RDV		**Vessel Packet Information Header**				
Communications identification (CID)	Report communication path quality	Vessel Report Status	UID (name of the vessel reporting)			
VPR		**Vessel Position Report**				
CID	UTC of position (hhmmss.ss)	Latitude (degrees:minutes.decimal)	Longitude (degrees:minutes.decimal)	Course over ground (degrees true)	Speed over ground (knots)	Vessel Report Status
GGA		**Global Positioning Fix Data**				
UTC of position (hhmmss.ss)	Latitude (degrees:minutes.decimal)	Latitude (N or S)	Longitude (degrees:minutes.decimal)	Longitude (E or W)	GPS Quality indicator	Horizontal dilution of position
VTG		**Track Made Good and Ground Speed**				
Track course over ground (degrees true)	Speed over ground (knots)					
ZDA		**Time and Date**				
Day (01 to 31)	Month (01 to 12)	Year				

Table 2.3: ADS Reporting Data Structure. After Ref. [5].

Data Element	Description
Vessel Name	
UTC	Time of sensor detection
Track ID	Tdbm unique track number
Sensor Track Number	Radar Track Number or DSC Number generated by reporting sensor
Course	degrees true
Speed	knots over ground
Latitude	degrees:minutes:seconds.decimal
Longitude	degrees:minutes:seconds.decimal
Tracking Status	Radar, SR, ADs, etc.
Size of Vessel	
Track Quality	GPS quality indicator, Radar track quality

Table 2.4: Data Elements for Tdbm Track History Recording

C. STANDARD ROUTES

Another form of track data that is available to the fusion algorithm is that generated by the Standard Route (SR) daemon. In the current implementation of VTS, when a vessel transits from an area of radar coverage, it is assigned to a SR. SRs are multisegmented routes through the AOR of the VTC. Each segment is geographically fixed to reflect a particular segment of a waterway or harbor. Predetermined vessel classes are assigned specific transit speeds along each segment to reflect their real world parameters. Once a vessel is assigned to an SR, the SR daemon updates the position of the vessel every 10 seconds in the Tdbm with a new track report calculated by projecting the vessel's course and speed ahead in time along the route defined [Ref. 7]. Initial data for projection is based on track information previously defined in the Tdbm. The SR is designed to automate the tracking of vessels as they transit from one coverage area to another.

It is possible to change the way SRs are implemented so that SR tracks are calculated and made available in the Tdbm whenever a radar or ADS track is present [Ref. 7]. This would have the advantage of providing continuity of platform number as a vessel was lost to the system from sensors with direct observation. The fusion algorithm would allow a unique platform number to be associated with the vessel through its transit of the VTS AOR. Two scenarios where this would be particularly useful are presented below in Figure 2.3. In Scenario One, no overlapping coverage is possible because of a large bridge blocking overlap by the two covering radars. Without SRs being produced, the system is unable to make the association that the vessel that emerges from the far side of the bridge is the same vessel that was lost and the track dropped on the near side. With SRs implemented, when the radar track is lost by the near side radar, the platform number will continue to be associated with the SR on which the vessel is assumed to be travelling. As the vessel emerges from the far side of the bridge and is acquired by the new radar, the new track will be associated with the SR track and therefore be promoted to the same platform as the SR thereby receiving the same platform number. With the fusion algorithm, this association process is automatic. Similarly in Scenario Two, once the radar track is lost, the platform track continues to be plotted based solely on the SR generated track. As the next radar acquires the vessel, the SR and new radar track are reassociated and the platform number is maintained.

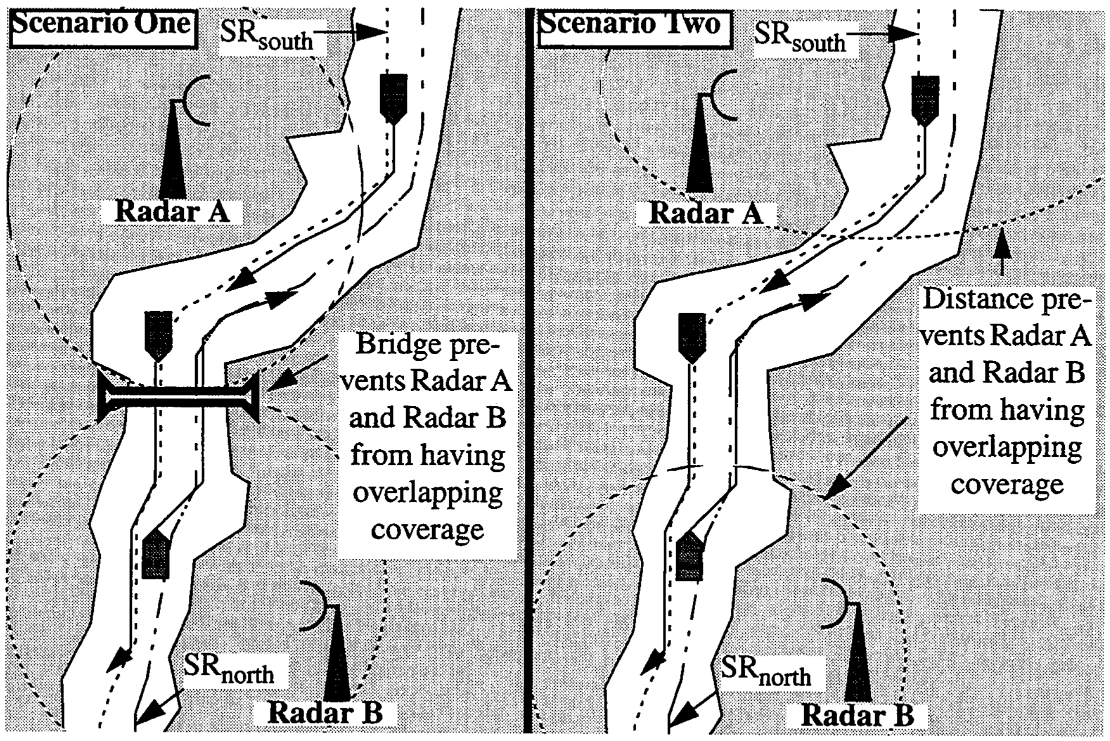

Scenario One
SR south
Radar A
Bridge prevents Radar A and Radar B from having overlapping coverage
SR north
Radar B
Scenario Two
SR south
Radar A
Distance prevents Radar A and Radar B from having overlapping coverage
SR north
Radar B

Figure 2.3. SR Utilization Possibilities

III. DATA FORMATTING AND PREPROCESSING

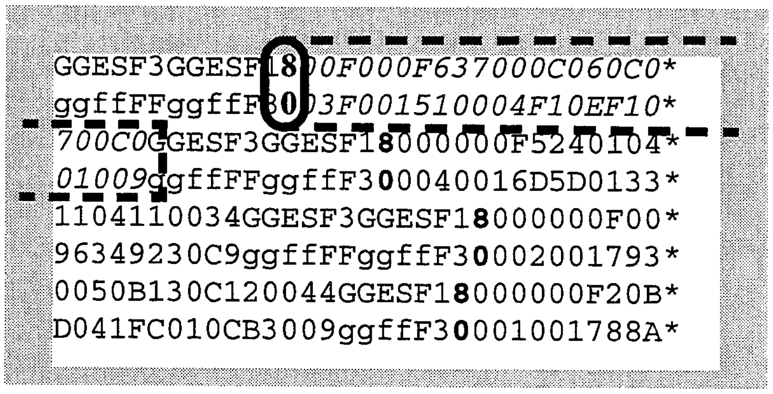
Once sensor tracks arrive at the central site, two things need to be done to prepare the tracks for the fusion algorithm. First, the relevant features of latitude, longitude, course, speed and size need to be obtained. Second, once the data has been prepared, it needs to be windowed to reduce the amount of data to be fused.

A. DATA FORMATTING

As discussed in the previous chapter, data for the fusion algorithm are provided from a number of sources or sensors. These include tracks generated by the Telephonics radar processor, ADS, and the internally generated SRs. These sources can be thought of as independent sensors providing information to be fused. In order for fusion to take place, however, the data need to be converted into the appropriate format. This requires extraction of the relevant features of latitude, longitude, course, speed and size. The preprocessing required is dependent of the type of sensor providing the data. The data format and preprocessing requirements for each sensor are now discussed.

1. Radar Tracks From Remote Site Processors

The data from the Telephonics RSPs are run length encoded in the SDLC format for transmission to the central site over either a T1 link or a microwave link. The actual data have the format as shown in Figure 3.1 when received by the central site. Illustrated is the header information (circled) and one complete track report (dashed box).



```
GGESF3GGESF1800F000F637000C060C0*
ggffFFggffF3003F001510004F10EF10*
700C0GGESF3GGESF18000000F5240104*
01009ggffFFggffF300040016D5D0133*
1104110034GGESF3GGESF18000000F00*
963492 30C9ggffFFggffF30002001793*
0050B130C120044GGESF18000000F20B*
D041FC010CB3009ggffF30001001788A*
```

Figure 3.1. Data Format from Telephonics RSP

The data are transformed into the format presented in the previous chapter in Table 2.2 by the JMCIS software. Once the track records have been converted into the suitable ASCII format, additional refinements are carried out to reduce the amount of data that needs to be retained and to extract the relevant features for fusion.

The following preprocessing was performed on the radar tracks to produce the output presented in Table 3.1:

- Latitude and longitude were calculated knowing the location of the reporting radar and the smoothed range and bearing of the vessel given in the radar track.
- Course and speed were directly carried over from the transmitted output of the RSP Kalman filter.
- The size feature was computed by calculating the depth of the target from the extent range and the breadth of the target knowing its extent azimuth and centroidal distance from the radar. Depth and breadth were then multiplied together to provide a size feature in m^2.
- Reporting site and site dependent track numbers were then concatenated together to form a unique track identifier in the system.
- GPS track time as recorded with the track was kept.

- Lost track (LT) was kept as an indicator.
- Track quality (TQ) was kept to provide a weighting factor when comparing the reliability of two tracks and to use as a modifier in the centroidal fusion process of the algorithm.

2. ADS Tracks

Minimal preprocessing is required with ADS generated tracks as the GPS latitude, longitude, course and speed are reported in the desired format. The size of the vessel can be determined through a query of the Oracle database where relevant details of ships in the AOR are kept.

3. SR Tracks

Similarly, little additional processing is required for SR generated tracks. The SR inherits the location, course, speed, and size of the vessel from the Tdbm when the SR track is started. The course and speed continue to be updated according to the SR attributes assigned to that leg for that vessel class.

Latitude	Longitude	Course	Speed	Size	Site & Track	Time	LT	TQ
(rad)	(rad)	(deg)	(kts)	(m^2)	STTT	(sec)		
0.0005954	-0.0007387	204.8	1.06	3620.2	1004	510.57	0	9
0.0001867	-0.0004711	17.8	5.12	537.1	1002	510.73	0	9
-0.0001032	-0.0002289	35.1	0.12	4152.3	1001	511.04	0	9
0.0005954	-0.0007387	200.2	1.06	3620.2	1004	513.12	0	9
0.0001872	-0.0004697	17.7	5.12	1143.3	1002	513.27	0	9
-0.0001033	-0.0002289	314.2	0.12	1135.6	1001	513.59	0	9
0.0005951	-0.0007389	199.6	1.06	3754.2	1004	515.67	0	9
0.0001879	-0.0004682	18.9	5.06	855.5	1002	515.82	0	9
-0.0001033	-0.0002289	337.1	0.12	2129.3	1001	516.14	0	9
0.0005958	-0.0007398	199.2	1.06	2416.3	1004	518.21	0	9
0.0001883	-0.0004668	19.2	5.12	924.7	1002	518.37	0	9
-0.0001033	-0.0002289	352.3	0.06	2235.7	1001	518.68	0	9
0.0005956	-0.0007401	197.2	1.06	6577.7	1004	520.75	0	9
0.0001891	-0.0004665	19.3	5.12	1138.1	1002	520.91	0	9
-0.0001033	-0.0002289	227.5	0.06	1561.5	1001	521.23	0	9
0.0005953	-0.0007403	199.2	1.06	3423.1	1004	523.30	0	9
0.0001893	-0.0004652	19.2	5.12	1277.5	1002	523.46	0	9

Table 3.1: Telephonics Data After Preprocessing

B. WINDOWING

Once the data has been preprocessed into the desired format for fusion, it needs to be windowed to reduce the amount of data required to be processed. The fusion algorithm can accept tracks from any sensor that can provide the necessary features. The following example uses radar track data obtained from the Telephonics MTE-2000 at the Electronics Engineering Center in Cape May, NJ to illustrate data windowing.

1. Select an N Second Window

Table 3.1 is the result of preprocessing the data set with **ReadData.m** (Appendix C, page 100). The gray boxes indicate the data selected by a six second window imposed on the data. Figure 3.2 graphically shows the process of applying a time window to the data arriving at the Tdbm.

In the simulation, a 15 second window was used. Windows can be of any length, but the optimum is to be only long enough to account for any latency of reporting due to communications and computation delays from sensor to Tdbm. The aim is to include the most recent reports from sensors providing overlapping coverage.

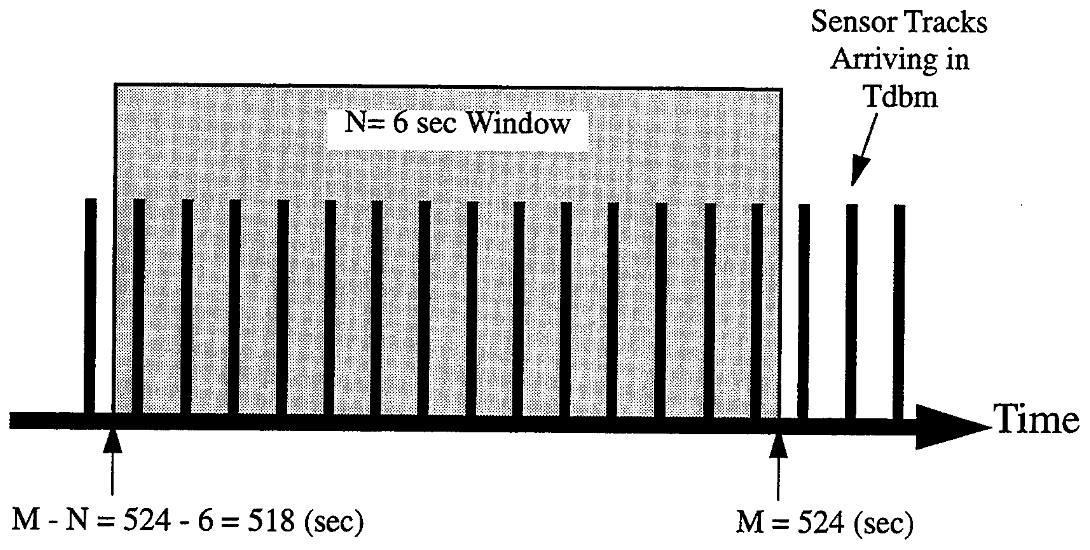


Figure 3.2. Illustration of an N Second Data Window

Table 3.2 was produced by applying **timeWindow.m** (Appendix A, page 55) to the data set in the previous table with N=6.

Latitude	Longitude	Crse	Speed	Size	Site & Track	Time	LT	TQ
(rad)	(rad)	(deg)	(kts)	(m^2)	STTT	(sec)		
0.0005958	-0.0007398	199.2	1.06	2416.3	1004	518.21	0	9
0.0001883	-0.0004668	19.2	5.12	924.7	1002	518.37	0	9
-0.0001033	-0.0002289	352.3	0.06	2235.7	1001	518.68	0	9
0.0005956	-0.0007401	197.2	1.06	6577.7	1004	520.75	0	9
0.0001891	-0.0004665	19.3	5.12	1138.1	1002	520.91	0	9
-0.0001033	-0.0002289	227.5	0.06	1561.5	1001	521.23	0	9
0.0005953	-0.0007403	199.2	1.06	3423.1	1004	523.30	0	9
0.0001893	-0.0004652	19.2	5.12	1277.5	1002	523.46	0	9

Table 3.2: Six Second Data Window

2. Select Most Recent Tracks

Figure 3.3 illustrates the selection of only the most recent tracks generated by each sensor. This produces a unique set of tracks that can be correlated to determine which ones should be associated with each other.

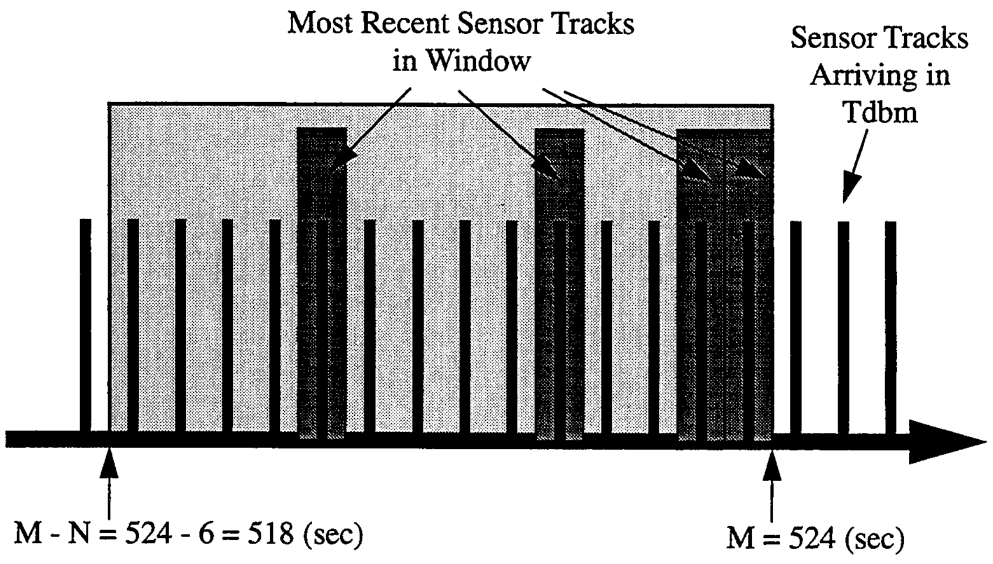

Most Recent Sensor Tracks in Window
Sensor Tracks Arriving in Tdbm
M - N = 524 - 6 = 518 (sec)
M = 524 (sec)

Figure 3.3. Most Recent Tracks

Table 3.3 shows the result of applying **mostRecentTracks.m** (Appendix A, page 56) to the data in the previous table. Only unique tracks are carried forward.

Latitude	Longitude	Crse	Speed	Size	Site & Track	Time	LT	TQ
(rad)	(rad)	(deg)	(kts)	(m^2)		(sec)		
-0.0001033	-0.0002289	227.5	0.06	1561.5	1001	521.23	0	9
0.0001893	-0.0004652	19.2	5.12	1277.5	1002	523.46	0	9
0.0005953	-0.0007403	199.2	1.06	3423.1	1004	523.30	0	9

Table 3.3: Most Recent Tracks in Data

In summary, sensor data received from different sensors are first converted into a unified format. Then, the track data is windowed to extract only the most recent tracks, which also results in data reduction. Once the data set has been reduced, data association can take place as presented in the next chapter.

IV. ALGORITHM

With the relevant features extracted and the most recent sensor observations isolated, the sensor tracks are now ready to be correlated and fused where necessary. This chapter first presents an overview of fuzzy association as it applies to fusion and then details its application to the VTS problem.

A. FUZZY ASSOCIATION FOR FUSION

The goal of the fusion algorithm is to combine or fuse tracks of the same vessel observed and reported to the system by different input devices whether from radar processors, ADS or some other mechanism. These fused tracks can then be associated with a unique platform identifier represented in the system by a unique platform number and a unique platform icon. The fuzzy membership is used to achieve this fusion. The membership function from fuzzy set theory provides a mechanism to measure correlation between observation or track pairs.

Data fusion is a process dealing with association, correlation and combination of data from multiple sources to achieve a refined position and identity estimation. [Ref. 3] The aim of the data fusion is to derive more information in the final result than is present in only a single source of information. The combination of multiple sensors has the added benefit of redundancy of reporting. The failure of a single sensor then is not critical for coverage of an area. In addition, multiple sensors provide improved spatial coverage of an area with improved resolution over that offered by a single sensor.

Data fusion is usually classified into three types: positional fusion, identity fusion and threat assessment. [Ref. 4] Positional fusion endeavors to determine an improved position estimate of a target by combining parametric data, such as azimuth, range, and range rate. Identity fusion uses known characteristics to determine the identity of a target. [Ref. 3] Threat assessment is the highest level of data fusion and is used for military or intelligence fusion systems to determine the meaning of the fused data from an adversarial point of view. [Ref. 4] The application of data fusion to JMCIS and VTS requires only positional fusion, and the method by which this is achieved is discussed further here.

B. POSITIONAL FUSION

Initial positional fusion is accomplished by the Adaptive Kalman filter tracker operating at each remote radar site. This is considered sensor level fusion. The proposed algorithm assumes that the sensor level fusion is being performed correctly and that valid tracks are being generated and sent to the central site for further processing. Central level positional fusion is performed at the central site with the aim of eliminating the redundancies in observations or tracks being generated by each of the sensor level fusion algorithms. These redundancies occur when there is overlapping coverage provided by sensors (i.e. two radars that cover the same waterway). Each radar gets returns on the target, starts a track and forwards the track information to the central site for display and historical record keeping.

As presented in previous chapters, additional redundant observations can result from the input of tracks from the Automated Dependent Surveillance (ADS) system [Ref. 5] or generated estimated positions (EPs) for vessels based on Standard Routes (SRs) generated by the Predictive Decision Support Aids (PDSA) [Ref. 2]. Each of these vessel observations appear in the Tdbm database [Ref. 6] along with a date/time stamp. Each of these sources of track information include sufficient information to generate the following attributes: position (latitude and longitude), course, speed and size.

Figure 4.1 shows how the data set is reduced as it flows through the algorithm.

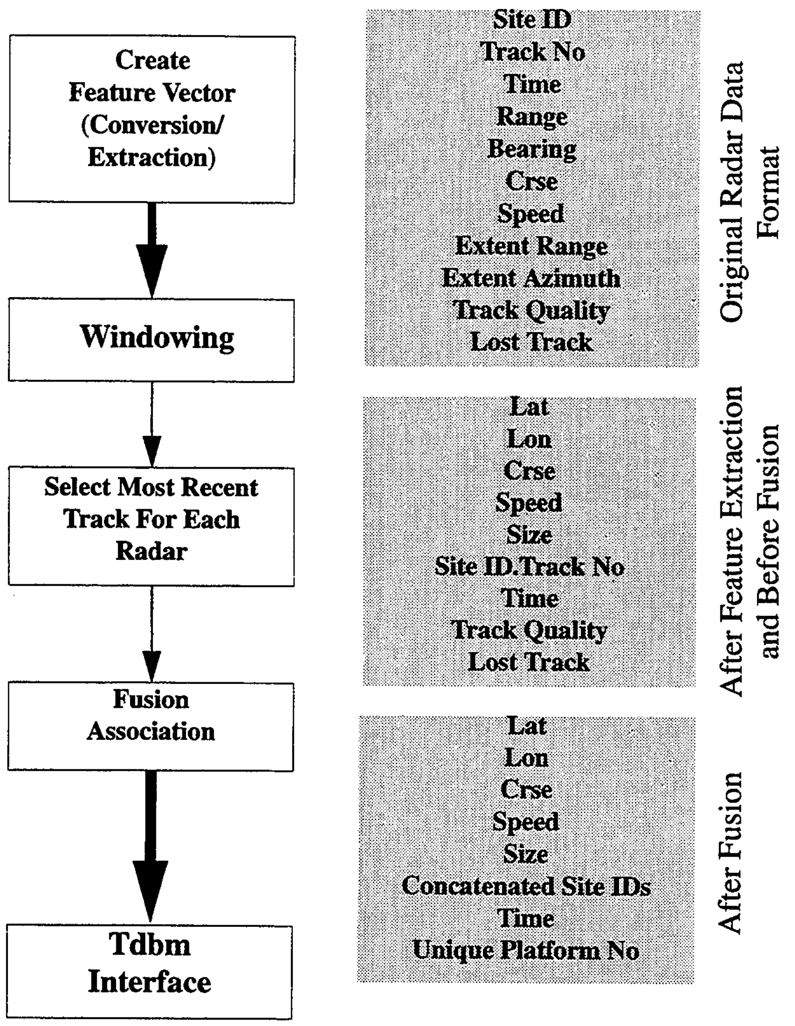


Figure 4.1. Fusion Algorithm Flow

The fuzzy association system takes these attributes and makes assignments of membership or similarity by correlation. This is accomplished as follows. Fuzzy set theory considers the partial membership of an object in a set. A membership function, $\mu(x)$, is used to grade the elements of a set in the range [0,1]. The grade of membership is a measure of the correlation of an object to a defined set. The closer the object is graded to one, the higher the membership of the object is in the set and the more compatible the object is with the set being considered.

Design of a fuzzy association system involves the following four steps: [Ref. 1]

- Determining the universe of discourse of inputs and outputs.
- Designing membership functions.
- Choosing fuzzy rules to relate the inputs and outputs.
- Determining a defuzzifying technique.

When comparing the latitudes of two separate radar tracks to see if they are similar a geometric membership can be constructed that takes into account the errors present in the system inherent to each remote site generating a track. A triangular shaped membership function as in Figure 4.2 is a good choice for a positional comparison because of the accuracy of radars in reporting the target position.

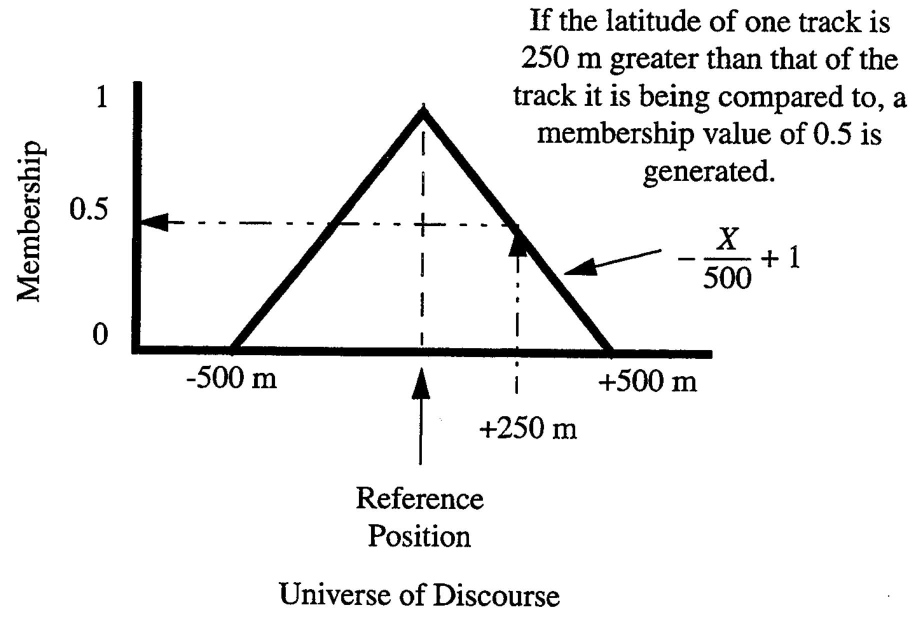


Figure 4.2. Position (Latitude/Longitude) Membership Function

In the example, the latitude given in one track is subtracted from the latitude given in another track held as the reference. The difference in latitude is used to determine the membership value.

Figure 4.3 shows the membership functions used in the algorithm.

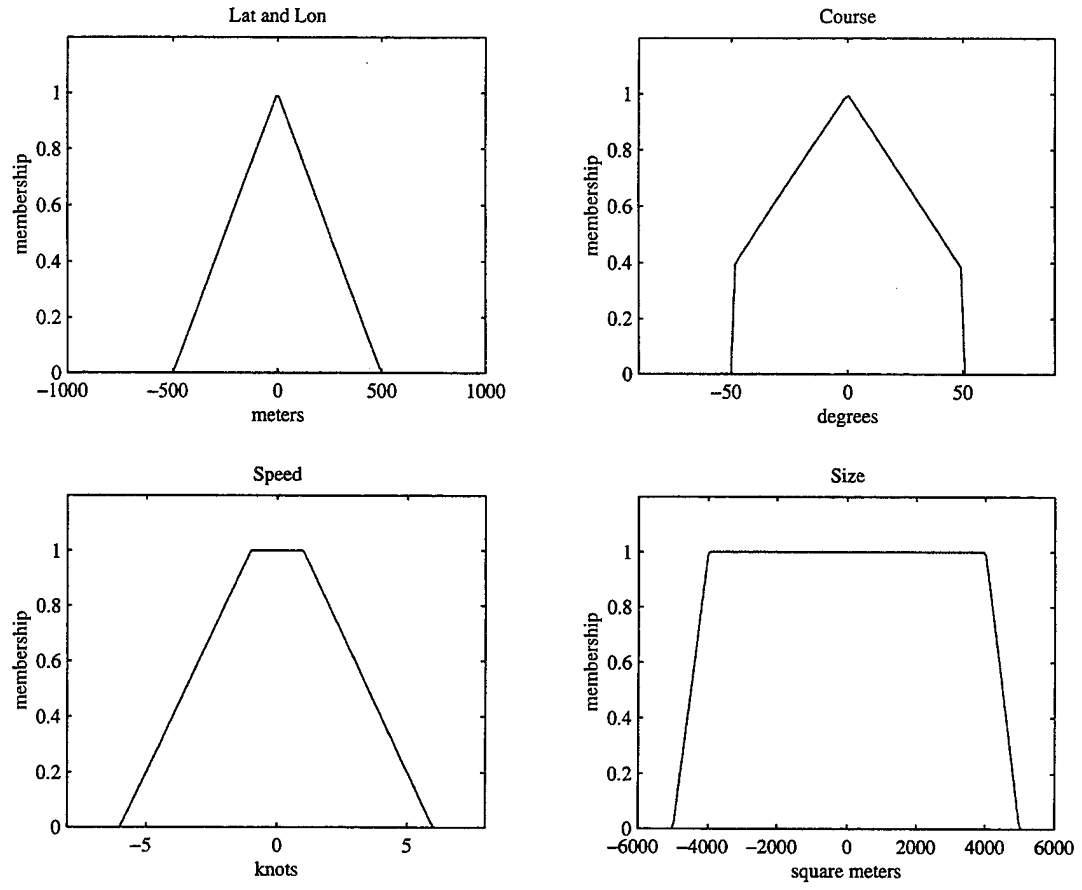


Figure 4.3. Membership Functions Used in Fuzzy Associative System

In general, the design of membership functions is based on the attributes inherent to those aspects being compared. Since both radar and ADS positions reported to the system are relatively accurate, the triangular membership function is appropriate. For other attributes where there is less accuracy such as in speed or size, broadening the roof of the membership function to include a greater range of values is valuable. It is also useful to truncate the membership function at a given value as in the case of the Course Membership Function creating a trapezoidal shape to allow a generous association within a reasonable range of values but not outside of a fixed range.

Next, in order to evaluate each of the membership values returned, a threshold needs to be established that reflects the physical limitations. In the case of the radar returns, a variable threshold is set that takes into account accuracy limitations of the radar dependent on the range of the target. Figure 4.4 graphically depicts the variable threshold employed in the simulation. The code that implements this function is **thresh.m** (Appendix A, page 60).

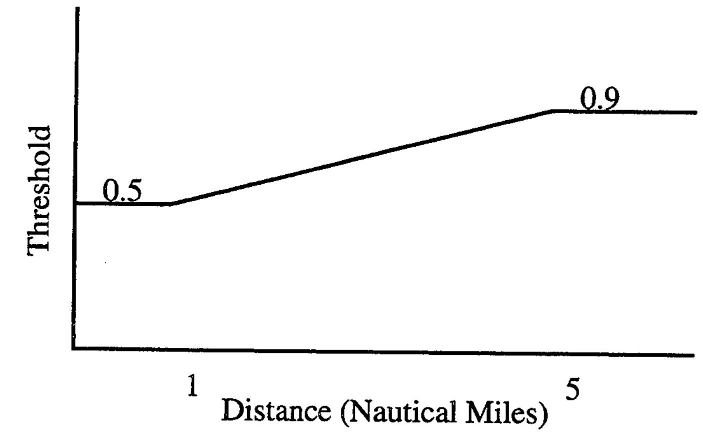


Figure 4.4. Variable Threshold Functions

Once all of the attributes for the track pair being assessed have been assigned membership values, they can be checked to see that they exceed the designated threshold. Each value is checked sequentially starting with latitude to ensure that it exceeds the threshold. If it does not, no further checks are made and association fails. This method has the advantage of computational efficiency. If all values exceed the assigned threshold, association is made as indicated by a binary output of '1' from the defuzzifier.

Figure 4.5 schematically shows the action of the fuzzy associative system. If the membership values, θ_i, all exceed the single threshold, ϕ, the two tracks would be associated. If the ϕ exceeds any of the θ_i, no association would be made.

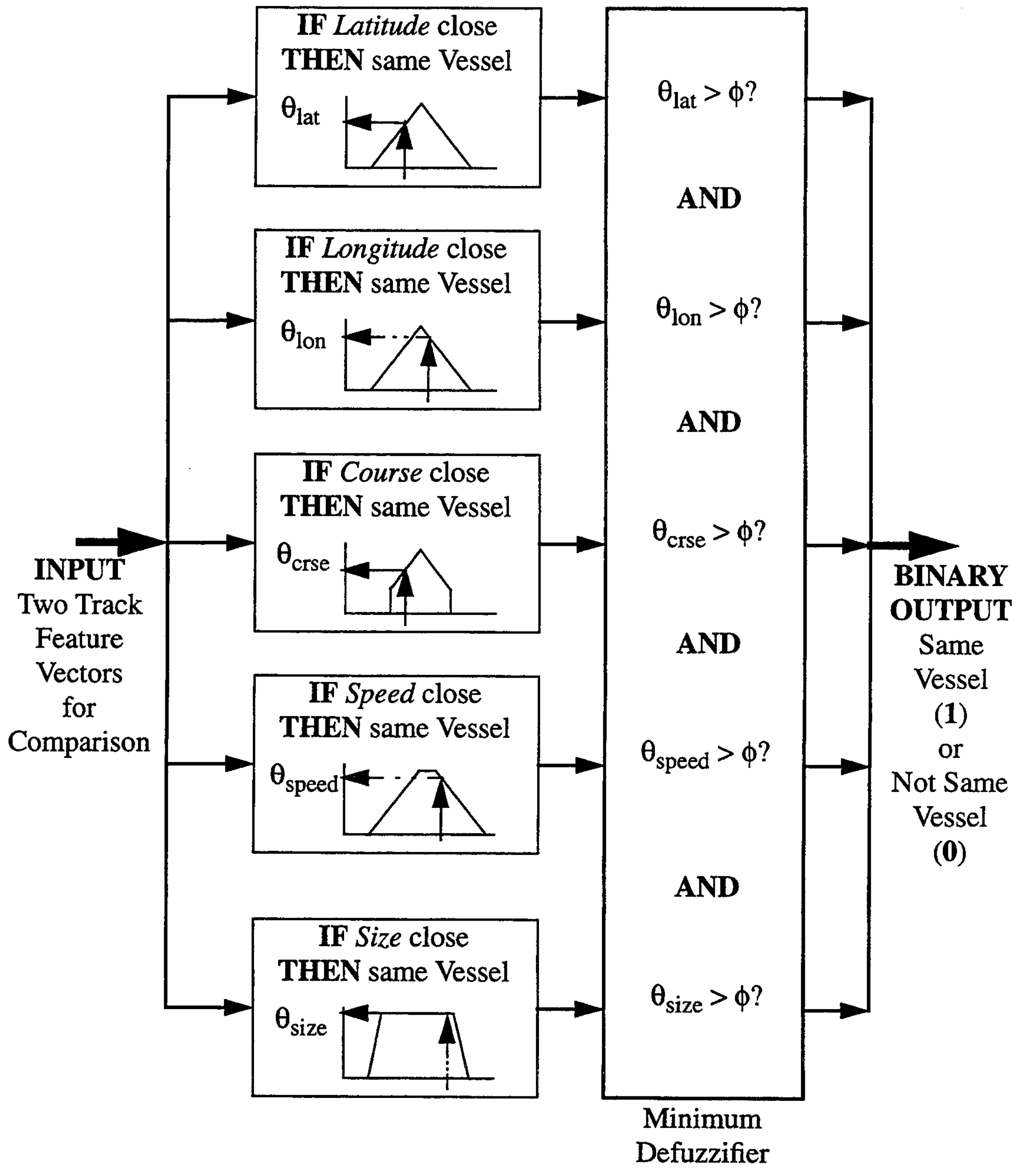

INPUT
Two Track Feature Vectors for Comparison
IF Latitude close THEN same Vessel
θ_{lat}
IF Longitude close THEN same Vessel
θ_{lon}
IF Course close THEN same Vessel
θ_{crse}
IF Speed close THEN same Vessel
θ_{speed}
IF Size close THEN same Vessel
θ_{size}
$\theta_{lat} > \phi$?
AND
$\theta_{lon} > \phi$?
AND
$\theta_{crse} > \phi$?
AND
$\theta_{speed} > \phi$?
AND
$\theta_{size} > \phi$?
Minimum Defuzzifier
BINARY OUTPUT
Same Vessel (1) or Not Same Vessel (0)

Figure 4.5. Fuzzy Associative System

In the fusion algorithm, the membership determination is made by the function **memshipWSize** (Appendix A, page 57) along with the variable threshold calculation performed by **thresh.m** (Appendix A, page 60). The membership values of all of these comparisons are stored in matrix *memshipValueOfTrk,* and the associated thresholds in *thresholdMatrix.*

The process of defuzzifying the results is accomplished with **associationMatrix.m** (Appendix A, page 61) where all the membership values are checked against their respective thresholds. The result is returned in *assocMatrix.*

The final step in the track fusion process is to fuse tracks that need to be fused based on the binary decisions stored in *assocMatrix.* This task is accomplished by **relateTracks.m** (Appendix A, page 62). The result is a single unified set of tracks representing a unique set of vessels present in the system in that time window. In the fused tracks, the original reporting sensor and its assigned track number are maintained for archival purposes as well as to assist in maintaining a unique platform number.

C. DATABASE FUSION

The data set is now ready to be used to update the Tdbm. The routine that performs this is **TdbmInterface.m** (Appendix A, page 64). The site and track number field is used to determine if this track being added is new to the system. If the search of the site and track number field in the Tdbm is successful, the associated platform number is appended to the track in question. If the search fails, a new platform track number is generated and the operator can be alerted to the new "unknown" track.

At this point the multilevel sensor fusion cycle is complete. The output of the various sensors have been related to each other, and the unified set has been related to the previous sets (the Tdbm). The data window can now be moved forward in time to gather in the next batch of sensor tracks and the process repeated. The next chapter will describe the simulation used to test the algorithm.

V. RESULTS

Attempts to acquire actual overlapping coverage data from the VTS system were not successful due to technical difficulties. As mentioned, only a limited amount of single coverage data was obtained. The data used in Chapter III to illustrate windowing was the result of applying the code detailed in Appendix C to raw single site data.

As an alternative to working on real data, a simulation was created to provide sensor tracks similar to the link tracks available in the Tdbm for the fusion algorithm to operate on. The area chosen for this simulation was the Upper Bay of New York Harbor whereby the Governor's Island and Bank Street radars provide overlapping coverage as depicted in Figure 5.1.

A. SIMULATION CONSTRUCTION

A Simulink simulation module was created to model vessel traffic transiting this area. Figure 5.1 through Figure 5.4 depicted the estimated tracks produced. Vessels were modeled with a speed of 10 knots and a turn rate of 45 degrees in three minutes. The simulation used Runge-Kutte 45 integration to compute the smoothed trajectory. The vessel position in terms of latitude and longitude was recorded at three second intervals of simulation time. This time interval reflected the actual expected update rate of track reporting to the VTC. The model is described in detail in Appendix D.

Four separate vessel tracks were generated and processed by the Multitarget Kalman filter presented in Appendix B. Before being processed by the filter, noise was added to the measurements by converting them to spherical coordinates and adding appropriate range and bearing variance to each set of measurements [Ref. 9]. The noise was modeled as follows:

- Range variance was based on 7 meter range bins and a uniform distribution;
- Bearing variance was based on taking 50 percent of the Half Power Beamwidth (HPBW) of the receiving radar and assuming a uniform distribution.

With noise added, each set of measurements was processed by the Kalman Filter. Filtering was performed with a q = 10 for slowly maneuvering targets [Ref. 10]. Filtering for each data set was performed from the perspective of the radar at Governor's Island Radar (Radar 1 in the simulation) and again from the perspective of Bank Street Radar (Radar 2).

The actual GPS survey locations for these sites were used to calculate measurement associations. The complete data sets were then truncated to provide a region of overlap only in the box defined by 39°N to 40.5°N and 02°W to 04°W. Although the real overlapping regions of coverage for these two radars is circular from the perspective of each radar, rectangular coverage served the purpose of illustrating where fusion should occur.

The data set at this point contained the variance present in the system for position (latitude and longitude), course and speed. The positional noise for each track can be seen in Figure 5.5 where the miss distances from the actual vessel trajectory are plotted. Course and speed were calculated using the mean of a three point moving average over one minute of simulation time. Figure 5.6 and Figure 5.7 show the output of the filter calculations for course and speed at each point. The boxes with the track numbers indicate the portions of the data set that were kept after truncating for geographic coverage.

In order to model the variance typical in the size feature as reported by radar processors, a statistical analysis was conducted on the limited data set provided by EECEN. Size was a difficult parameter to accurately model because of its dependence on not only the aspect of the vessel presented and the distance of the vessel from the reporting radar, but also the variance in range and bearing variance of the observing radar. From the analysis, it was determined that to achieve roughly the same distribution, the size could be modeled with a normal distribution out to one standard deviation below an arbitrary mean size and two standard deviations above. The size feature was randomized accordingly. Figure 5.8 shows the histograms of the size features used for each vessel.

The resulting tracks were then combined into one unified track table representing sensor tracks in the Tdbm. Figure 5.9 shows the plots of each of the tracks. The fusion algorithm was then fed tracks as determined by a sliding 15 second time window moving at three second increments. An animation was generated to monitor the progress of the fusion algorithm. Figure 5.10 through Figure 5.12 show snapshots of the output of the algorithm plotted at every fifth (15 second) point. Where fused tracks have been plotted, the originating sites and tracks numbers are shown concatenated together.

The output of the algorithm was appended to the Tdbm at each iteration. Independent redundant databases of tracks fused and tracks not fused were generated to simplify performance analysis of the algorithm.

B. RESULTS

The algorithm performed correctly under all test scenarios. The test scenarios were as follows.

- Vessels moving in and out of the overlapping cover area (Figure 5.10 and Figure 5.12).

- Vessels crossing within multiple coverage area with closest point of approach of 100 meters (Figure 5.11).

- Two vessels of differing deterministic size with the same location, course and speed.

Plots of the resulting fused and not fused tracks are presented in Figure 5.13 and, in a magnified view of the overlapping region, Figure 5.14. The following results were observed:

- The algorithm correctly fused all tracks within overlap region.

- The fusion algorithm was able to discriminate vessels with identical position, course and speed but of different size when the size feature was deterministic.

- The algorithm was also able to correctly fuse tracks with similar features within single coverage areas.

One of the key observations was the effect of the design of the individual membership functions. If the range of the membership function was not sufficiently broad, particularly in the case of the stochastic size parameter, the decision to fuse two tracks was not made.

In summary, the algorithm correctly identified unique tracks and associated a unique platform number with them which remained consistently associated as the vessel transited through multiple coverage areas. The algorithm did fuse N tracks correctly where 2N duplicate tracks were present in the system. Figure 5.15 shows the resultant unique platform tracks generated and stored in the Tdbm.

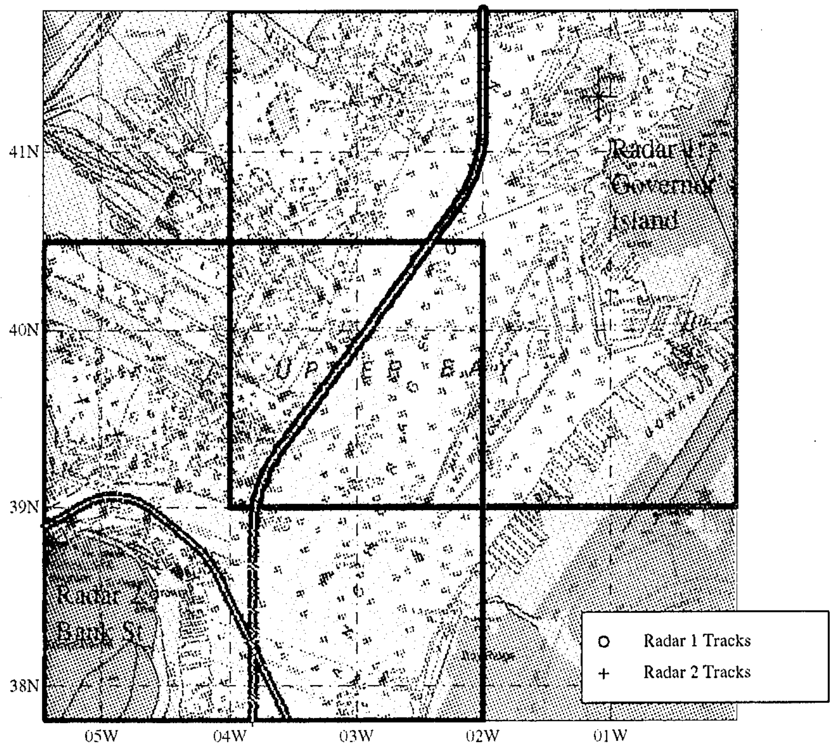


Figure 5.1. Estimated Tracks For Vessels A and B

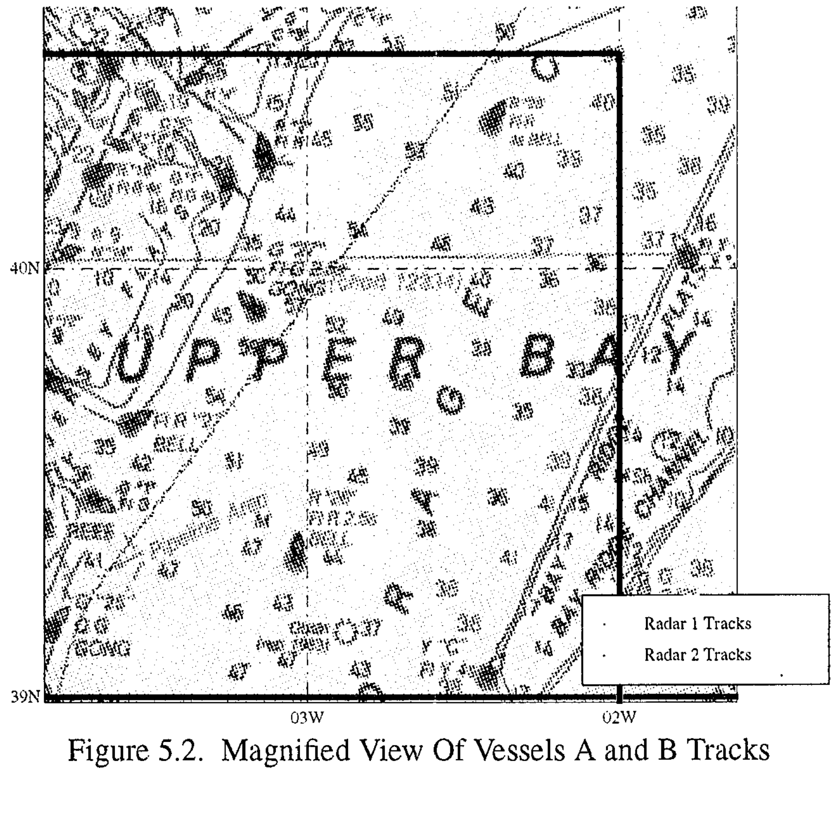


Figure 5.2. Magnified View Of Vessels A and B Tracks

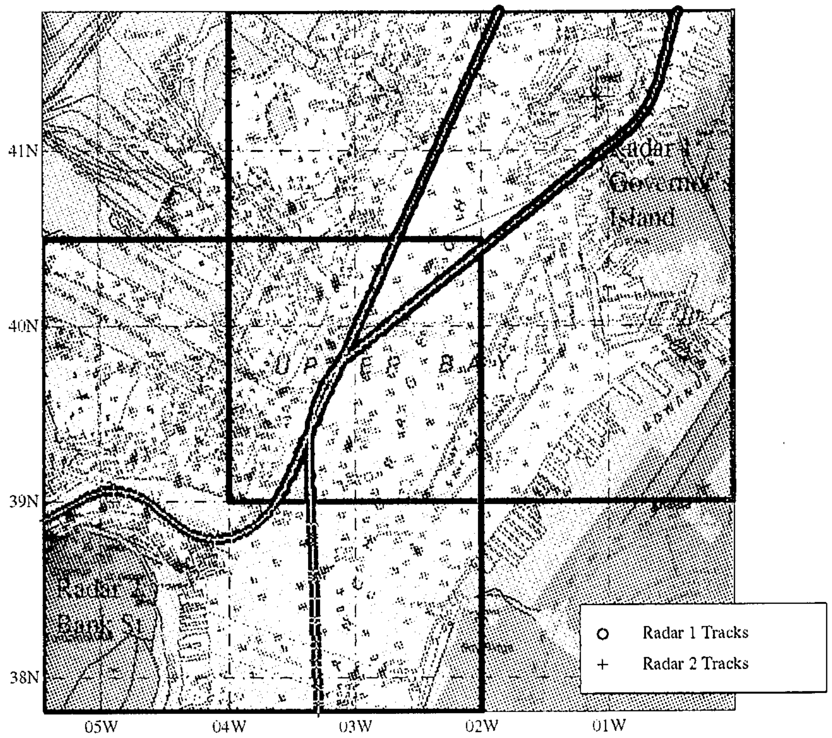


Figure 5.3. Estimated Tracks For Vessels C and D

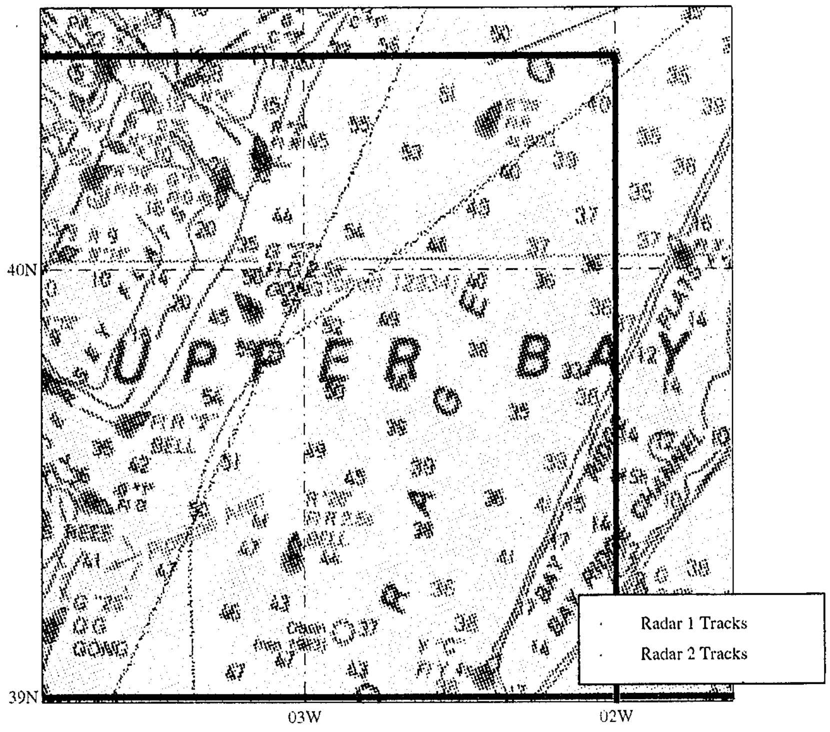
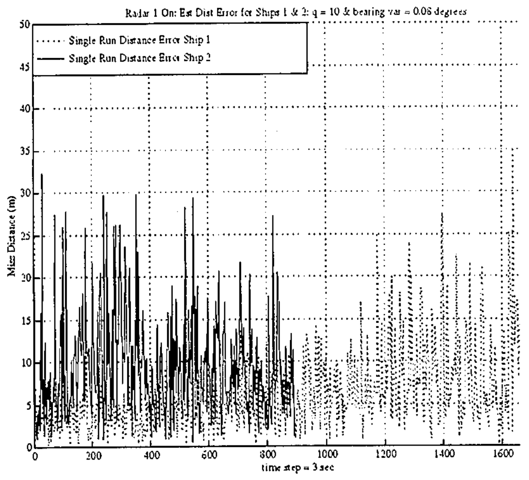
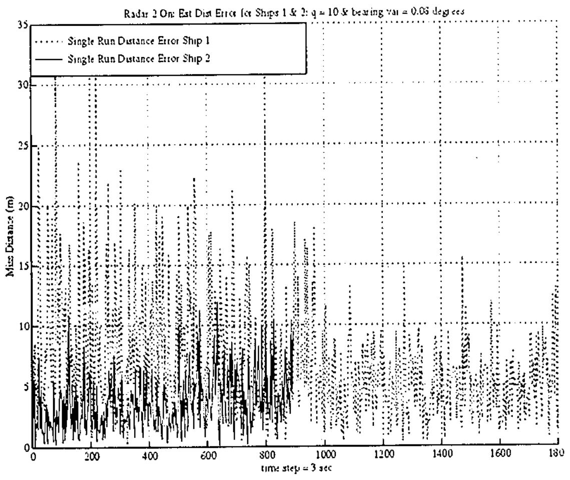


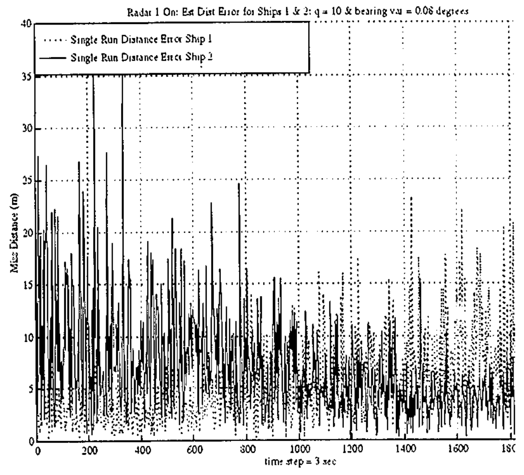
Figure 5.4. Magnified View Of Vessels C and D Tracks



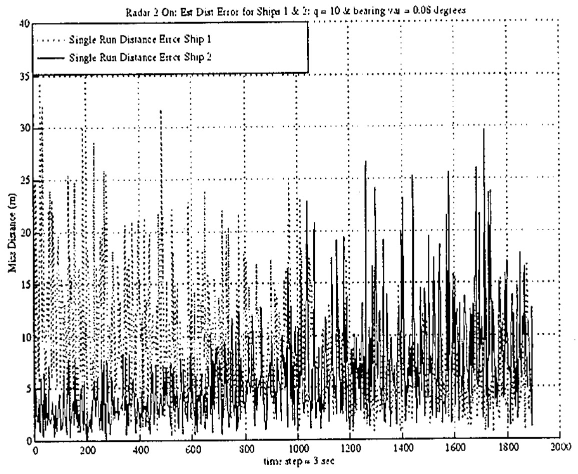
Kalman Output of Radar 1
on Vessels A and B



Kalman Output of Radar 2
on Vessels A and B

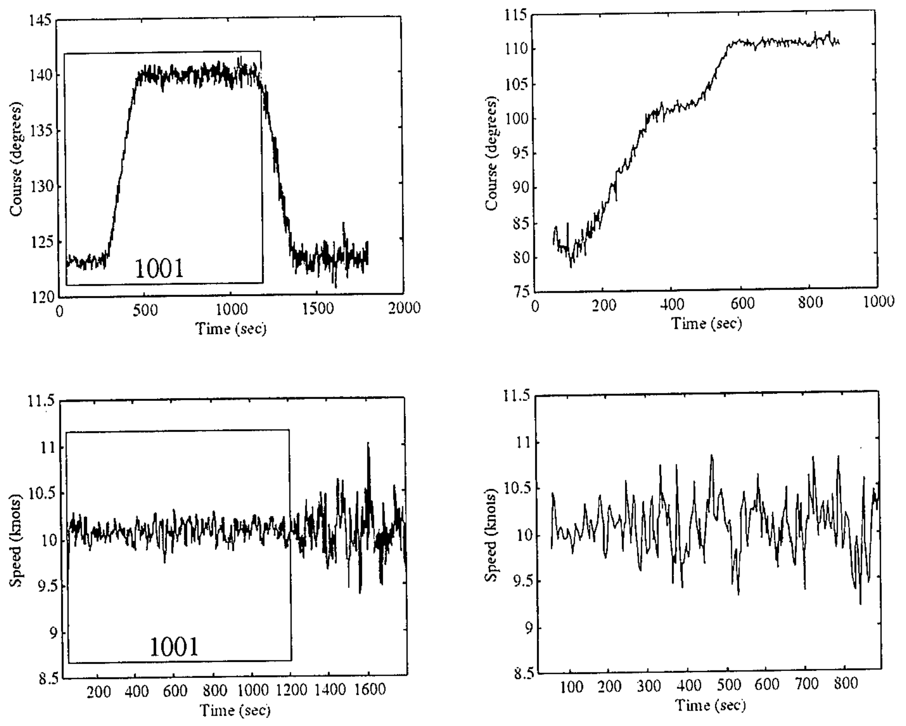


Kalman Output of Radar 1
on Vessels C and D

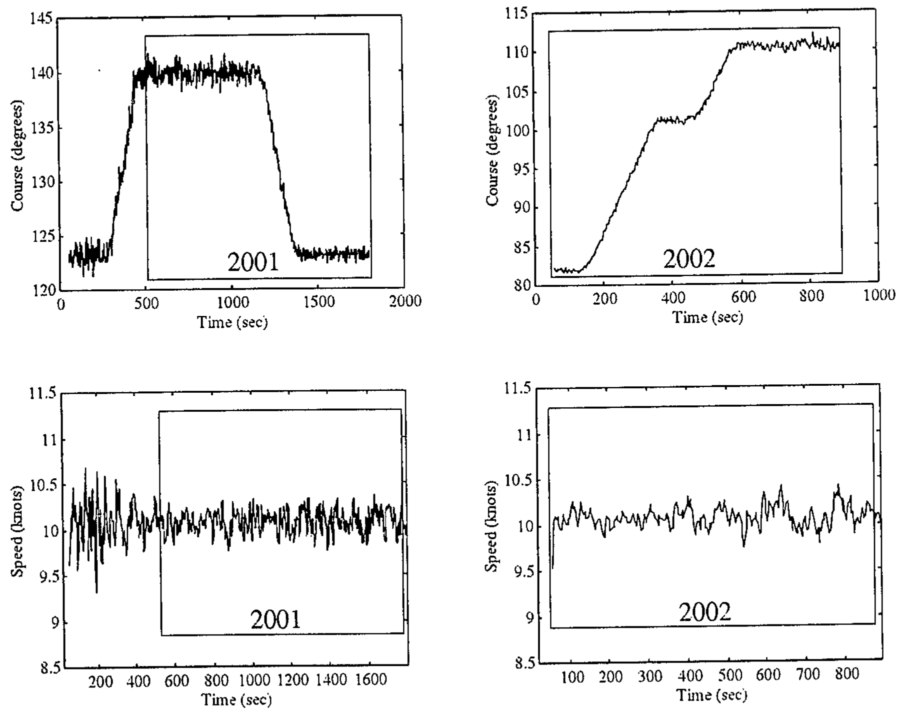


Kalman Output of Radar 2
on Vessels C and D

Figure 5.5. Miss Distances from Actual Track

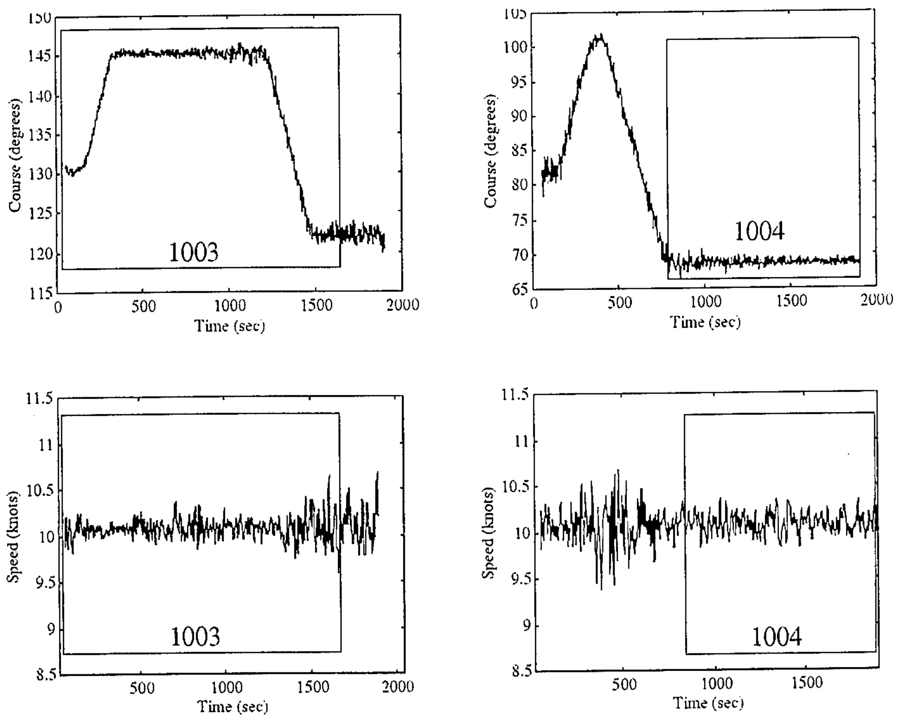


Calculated at Radar 1

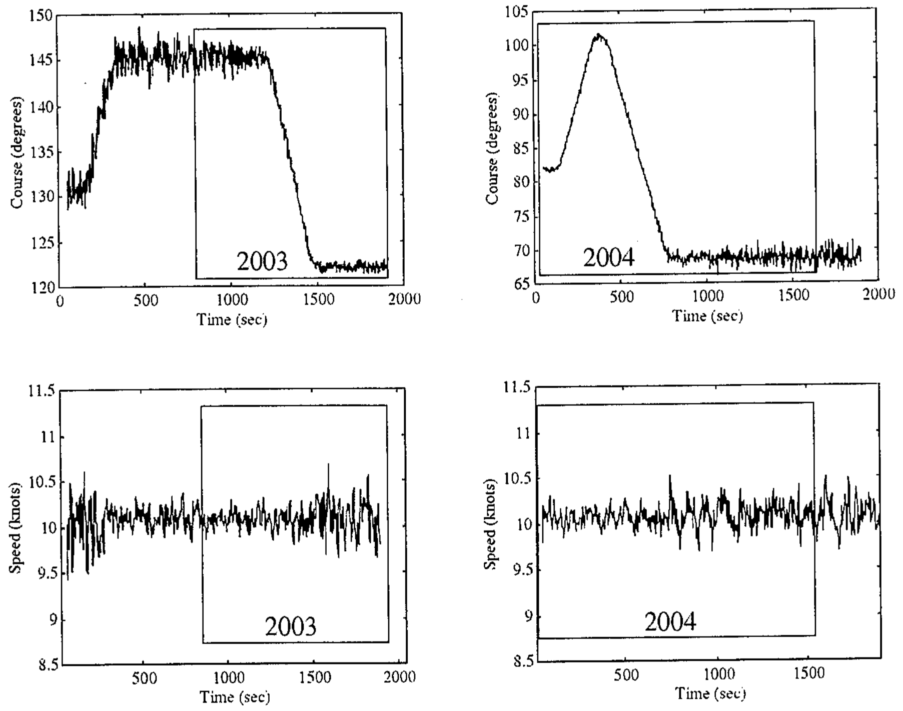


Calculated at Radar 2

Figure 5.6. Calculated Course and Speed of Vessels A and B



Calculated at Radar 1



Calculated at Radar 2

Figure 5.7. Calculated Course and Speed of Vessels C and D

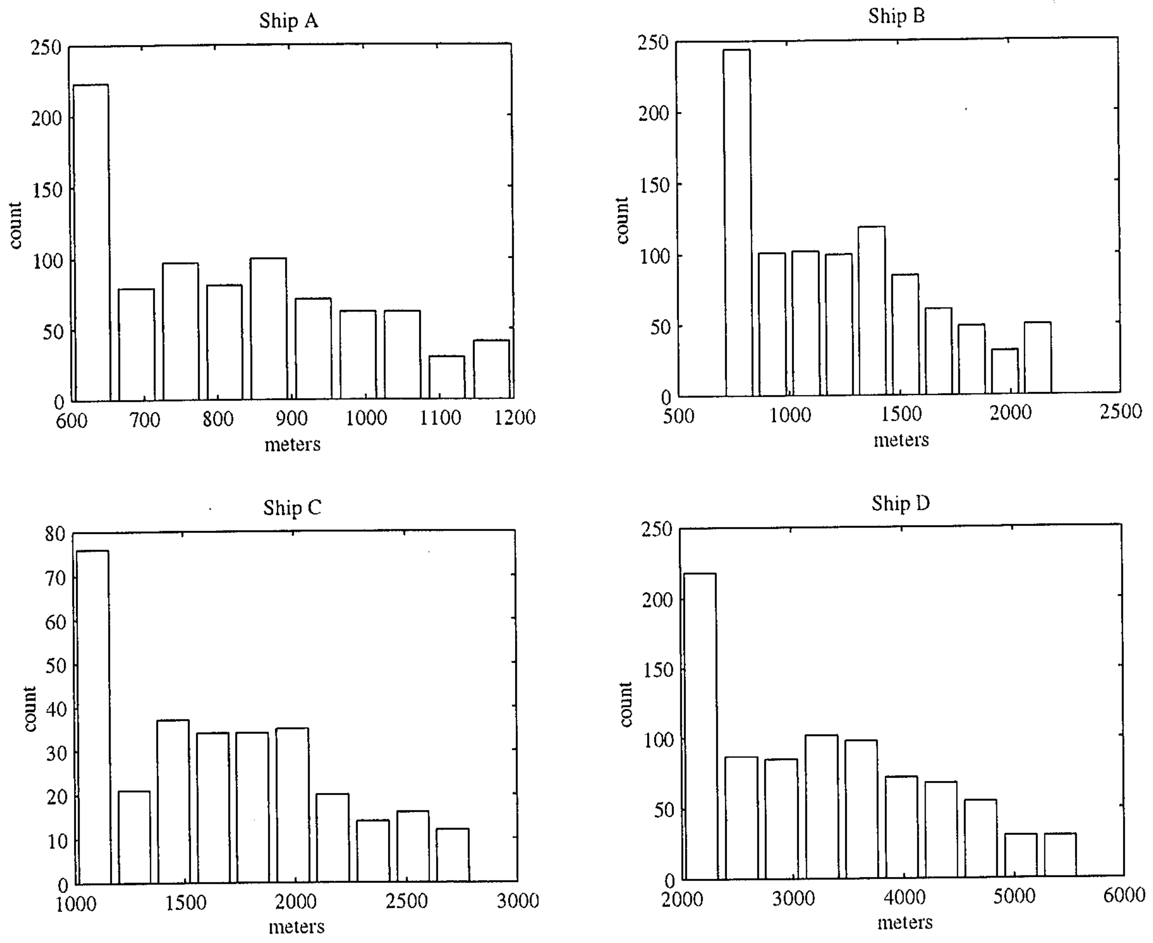
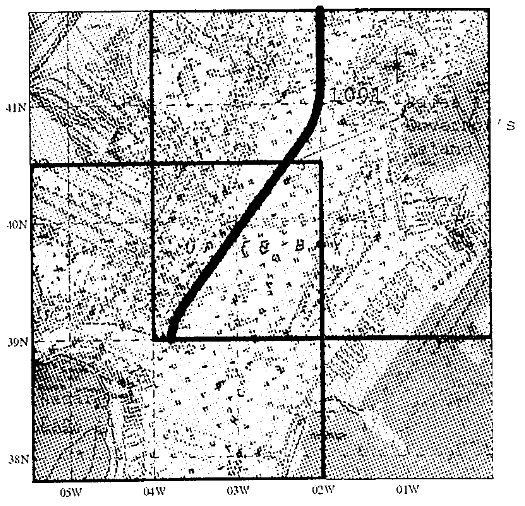
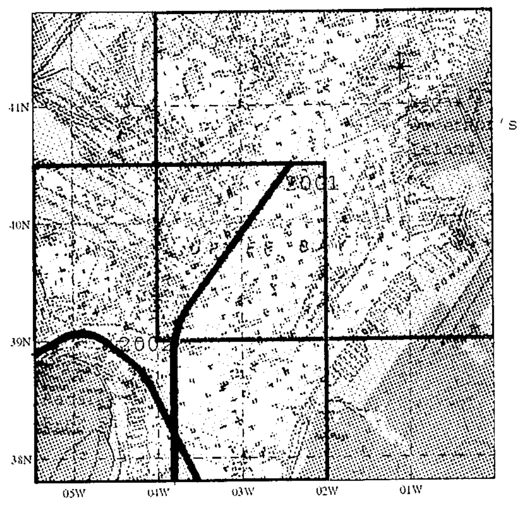


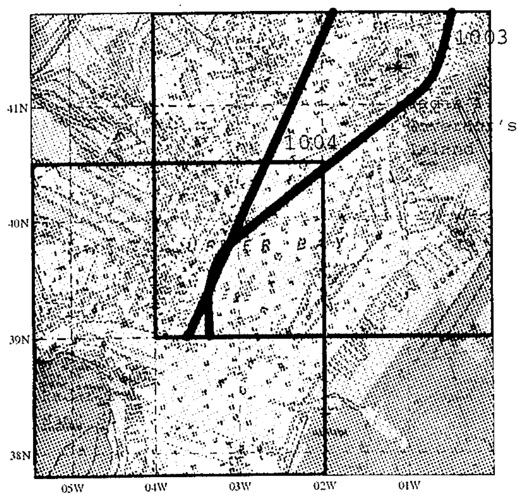
Figure 5.8. Size Feature Histograms



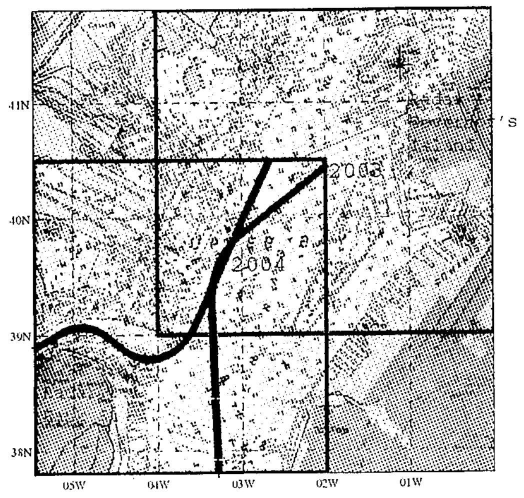
Track 1001



Tracks 2001 and 2002

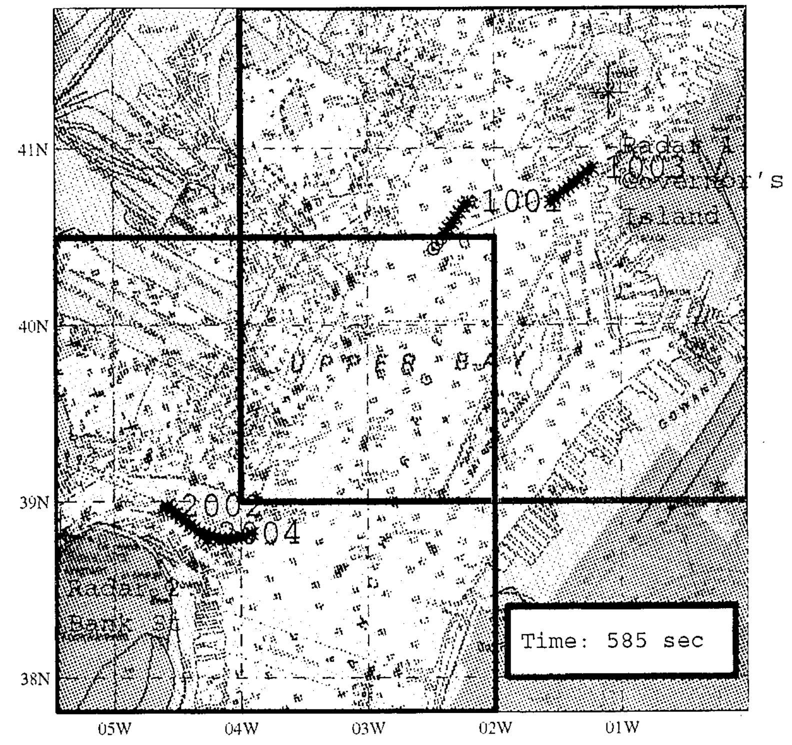


Tracks 1003 and 1004



Tracks 2003 and 2004

Figure 5.9. Tracks Input to Fusion Algorithm



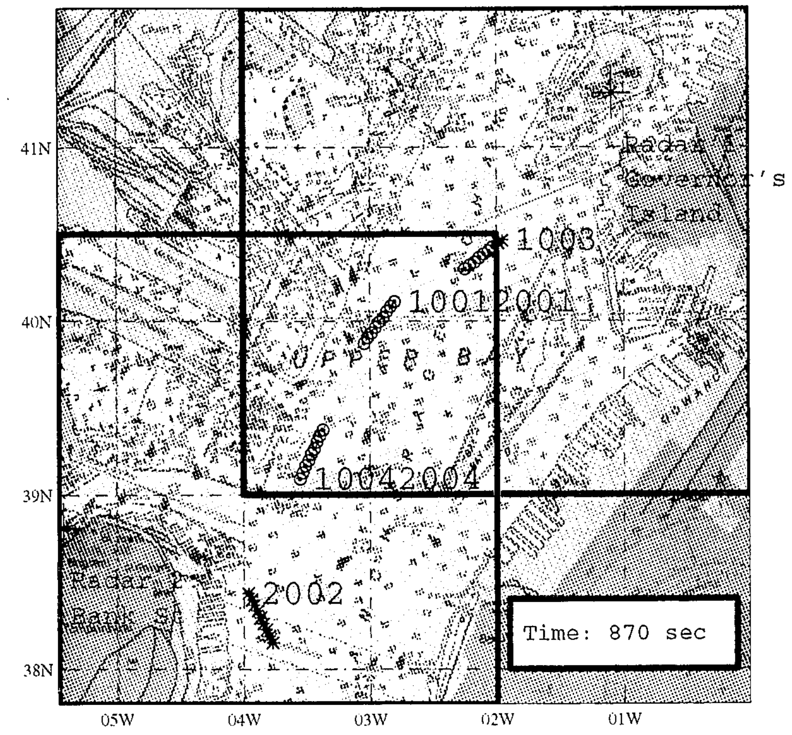
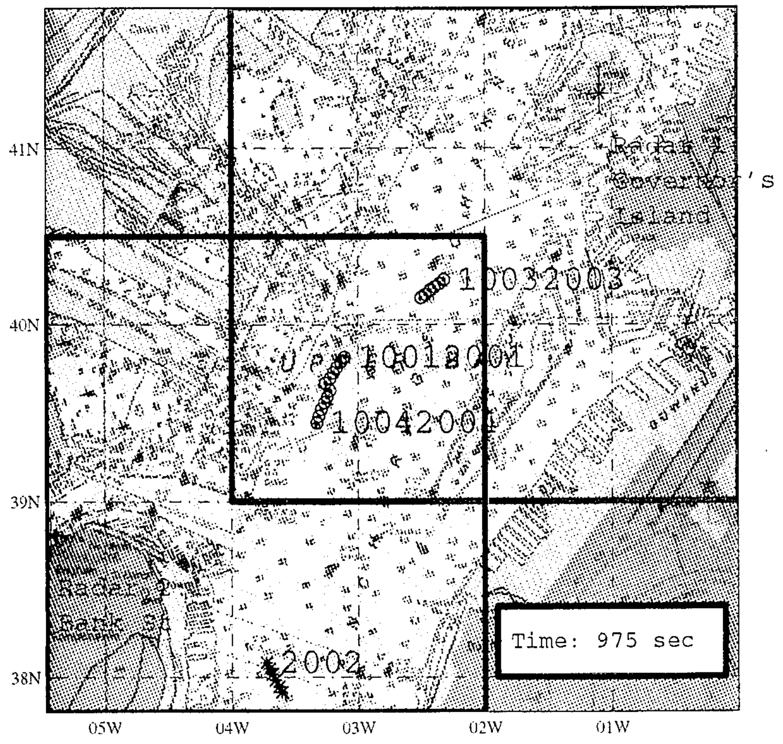


Figure 5.10. Fusion Algorithm Output at Time 585 and 870



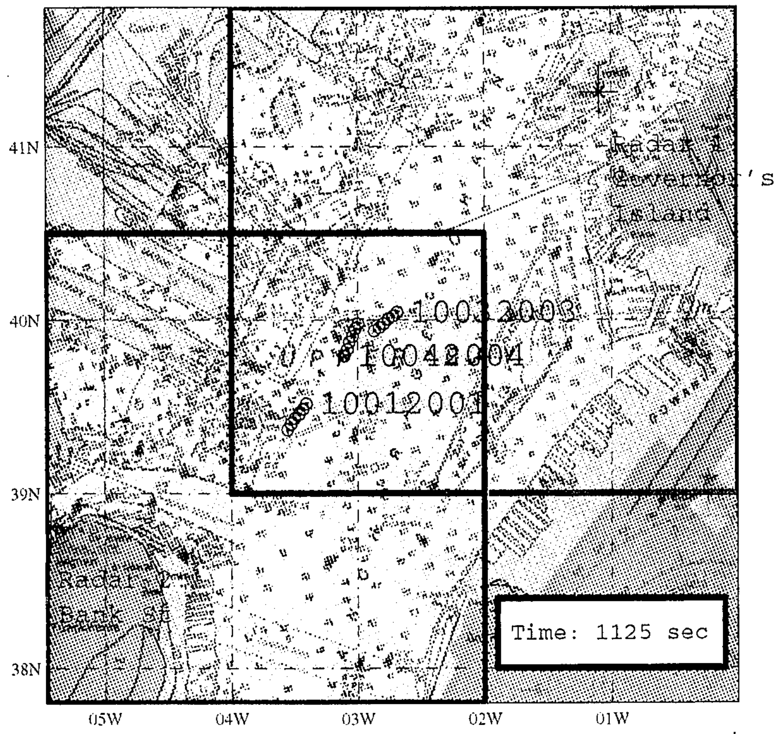


Figure 5.11. Fusion Algorithm Output at Time 975 and 1125

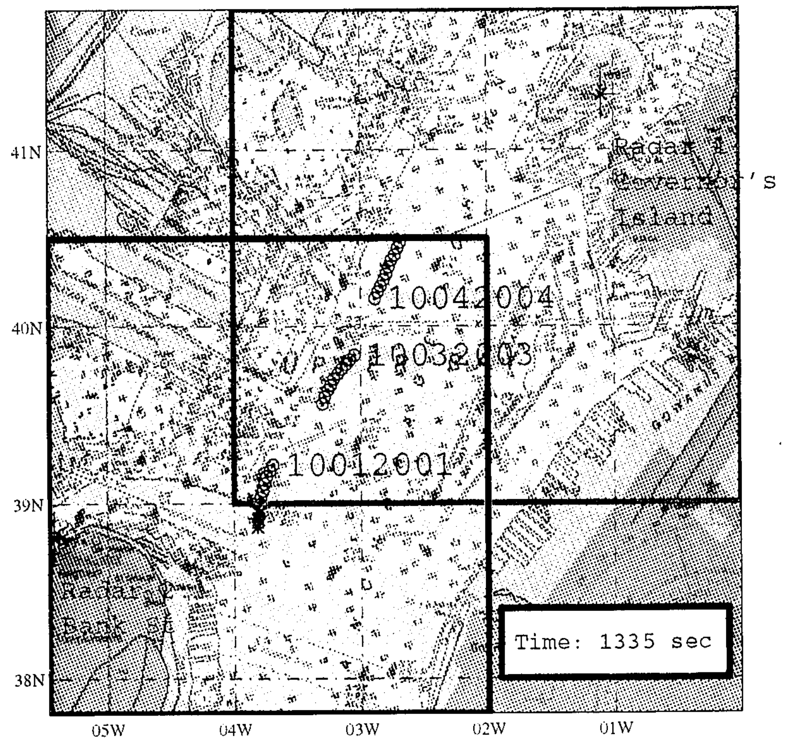


Figure 5.12. Fusion Algorithm Output at Time 1335

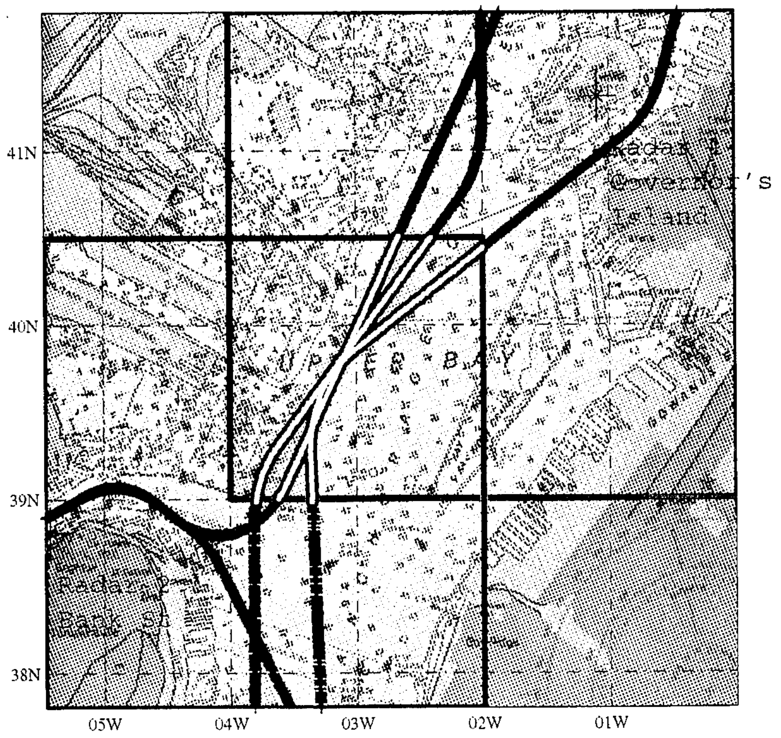


Figure 5.13. Post-Fusion: Fused (magenta) and Not Fused (cyan) Tracks

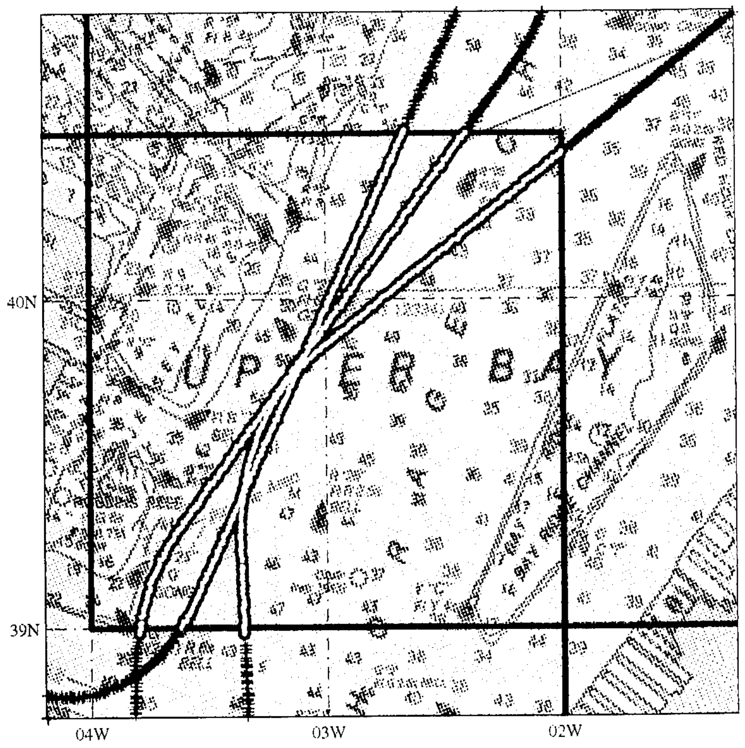


Figure 5.14. Post-Fusion: Fused Tracks in Overlap Region

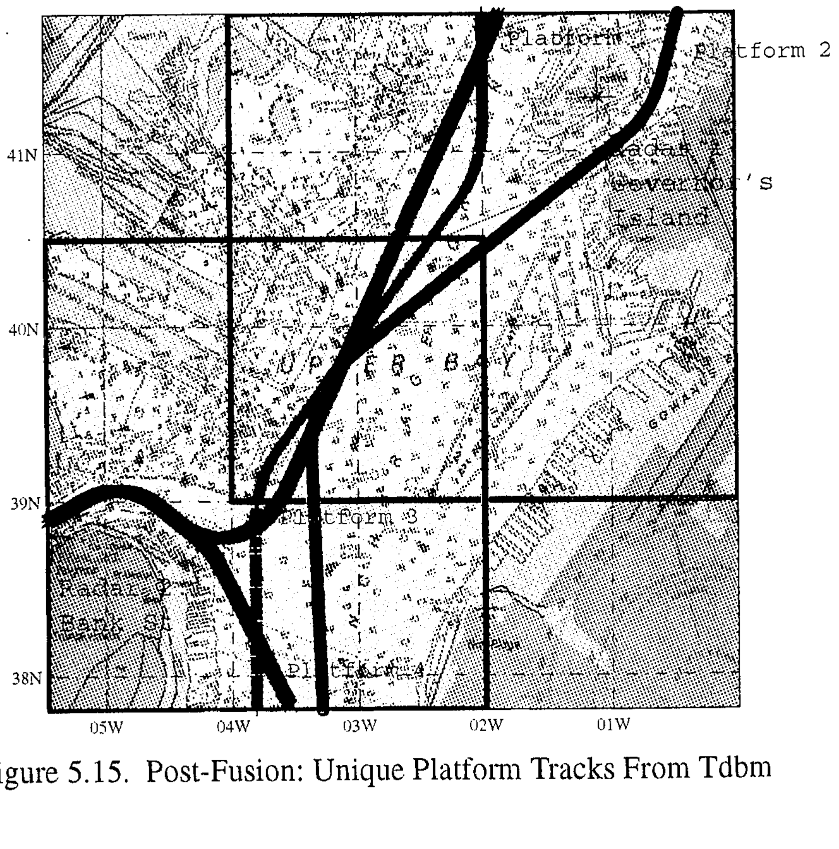


Figure 5.15. Post-Fusion: Unique Platform Tracks From Tdbm

VI. CONCLUSION

A. DISCUSSION OF RESULTS

The algorithm performed as expected, fusing tracks that represented multiple coverage of single vessels to produce a unified set of platform tracks in the Tdbm. This set represents unique vessels reported to the system. Variance in the parameters of each of the features strongly effects the range and shape of each of the membership functions used to determine association. The more accurately known the variance of a specific feature, the more precise the design of the membership function can be. The result is more accurate association of tracks. The fusion algorithm was computationally efficient and could accurately discriminate vessels. The algorithm could also handle an arbitrary number of vessels from an arbitrary number of sensors of arbitrary type as long as they were capable of providing some of the five features used for fusion. The algorithm can be easily modified to turn off the evaluation of specified features if those features are not present in the reported tracks. The algorithm can also be modified to add additional features.

B. SUGGESTIONS FOR FURTHER DEVELOPMENT:

1. SR Implementation

The mechanism by which SRs are generated and stored within the Tdbm can be modified to allow them to be computed whenever a radar track or ADS track is present. This would allow continuity of platform number and tracking whenever there is a gap in physical sensor coverage of the vessel.

2. Integration into JMCIS

The fusion algorithm was written with the intention of future integration into JMCIS and the VTS implementation. Most of the preprocessing requirements are already implemented in JMCIS and VTS. The primary additional code required is the ability to calculate a size feature. This involves some simple calculations or a call to the VTS Oracle database. As the fusion algorithm consists mainly of IF-THEN-ELSE constructs, it will benefit from recoding into the C language. Interface with the Tdbm to update with new sensor track information will be straightforward using the calls defined in the Tdbm Application Programmers Interface [Ref. 6].

3. Membership Function Design

When real sensor track data is available, statistical analysis of each of the features should be conducted. The membership functions presented could then be either validated or modified to ensure proper track association.

C. SUMMARY

This thesis is a continuation of a previous thesis [Ref. 1] to construct a data fusion algorithm capable of fusing data from multiple sources to produce a unique track table that represents a unique set of vessels transiting the system each designated with a unique identifier, the platform number. The United States Coast Guard will be able to implement this algorithm at their Vessel Traffic Centers to reduce the workload on both the system and the operators. This will allow for the operators to focus on the flow of traffic with less distraction, the implementation of automatic alarms (collision, etc.), and the keeping of a more accurate historical database than is presently kept.

APPENDIX A. ALGORITHM CODE

This appendix lists the following code for the fusion algorithm:

- **FusionAlgorithm.m** on page 52..
- **timeWindow.m** on page 55.
- **mostRecentTracks.m** on page 56.
- **memshipWSize** on page 57.
- **thresh.m** on page 60.
- **associationMatrix.m** on page 61.
- **relateTracks.m** on page 62.
- **TdbmInterface.m** on page 64.
- **randomSize.m** on page 66.
- **showChart1.m** on page 69.
- **inputPlots.m** on page 72.
- **animationPlots.m** on page 74.
- **outputPlots.m** on page 76.
- **plotMembership.m** on page 78.

FusionAlgorithm.m

```
%
%       FusionAlgorithm.m
%
%       Written by: Major Ian Glenn
%
%       Created:                20 Oct 95
%       Modified:               22 Nov 95
%
%       Input:
%       I1                              - Track data from Telephonics Remote Processor
%       I2                              - Simulation data
%
%       Output:
%       O1                              - Tdbm
%
%
%       Design:
%                               load the simulation/test data and run the Fusion algorithm on it
%
%       Calls:
%                               For Preprocessing of Telephonics Radar Data
%
%               ReadData.m
%               tracks3.m
%               hex2bstr.m
%               printPretty.m
%               simpleCoordConv.m
%               naut2mathRad.m
%               rads2DMS.m
%
%                               For Randomizing the Size Feature of Simulation Data
%
%               randomSize.m
%
%                               For Fusion of Track Data
%
%               timeWindow.m
%               mostRecentTracks.m
%               memshipWSize.m
%               associationMatrix.m
%               relateTracks.m
%               TdbmInterface.m
%
%                               For Plotting Results
%               inputPlots.m
%               animationPlots.m
%               outputPlots.m

clear Tdbm
plotNotFusedTrks = []
plotFusedTrks =[]
PlotMe          =2
RandomSize  =1          %       1 if want to randomize the size input
MovieOn         =0

Preprocess =0   % If using real preprocessed data set to 1
Simulation = 1  % OR if using simulation data set this to 1

if Preprocess == 1
        ReadData
end

if Simulation == 1% Prepare simulation data
```

```
        if exist('TrackTableRdr1') == 0
                load TrackTableGIa.mat;
        end

        if exist('TrackTableRdr2') == 0
                load TrackTableBanka.mat;
                %  delay the start of vessel track 2002 by 2 minutes or 120 seconds
                TrackTableRdr2(find(TrackTableRdr2(:,6)==2002),7) = ...
                        TrackTableRdr2(find(TrackTableRdr2(:,6)==2002),7)+120;
        end

        if exist('TrackTableRdr1b') == 0
                load TrackTableGIb.mat;
        end
        if exist('TrackTableRdr2b') == 0
                load TrackTableBankb.mat;
        end
        if RandomSize == 1
                if exist('ObsnMatrixRandSize') == 0
                        randomSize
                end

                ObsnMatrix = ObsnMatrixRandSize;
        else
                ObsnMatrix = [TrackTableRdr1;TrackTableRdr2;TrackTableRdr1b;TrackTableRdr2b];
        end

        if PlotMe >= 3
                inputPlots(TrackTableRdr1,TrackTableRdr2,TrackTableRdr1b,...
                        TrackTableRdr2b, ObsnMatrix)
        end     % end PlotMe

end     % end Simulation

        if PlotMe >= 1
                figure(6)
                showChart1
        end     % end PlotMe

startTime =min(ObsnMatrix(:,7)) % start at the earliest time in ObsnMatrix
windowSize = 15;                % Number of seconds to include in window
windowStart = startTime +3 % Take first 3 seconds of sensor data
endTime =               max(ObsnMatrix(:,7))% run to end of ObsnMatrix


for  window = windowStart :3:endTime% Slide the window in 3 second increments

        %*********** Windowing *************
        %       Once the data is available, it is windowed to extract the relevant tracks
        %       In a specified time interval by timeWindow.m

        WindowedObsns = timeWindow(ObsnMatrix,window,windowSize);

        % Note: the reduced matrix may be seen with printPretty(WindowedObsns)
        %
        %       The next step is to extract the most recent observation appearing in the
        %       windowed data.

        [MostRecentTrks] =  mostRecentTracks(WindowedObsns);

        %*********** Fusion *************
        %        Membership
                %       The track data is now ready to be fused together
                %       The first step is to compare each track with every other
                %       track to determine it's threshold and membership
                %       with the functions memshipWSize.m
                %       and thresh.m
                % Note the values of 0.9 and 0.5 being used for the variable
                %       threshold

                [memshipValueOfTrk, thresholdMatrix] = memshipWSize(MostRecentTrks, 0.9, 0.5);

        %       Association
                %       With the information in memshipValueOfTrk and thresholdMatrix
                %       the association of tracks can take place using
                %       associationMatrix.m

                assocMatrix = associationMatrix(memshipValueOfTrk, thresholdMatrix);

        %       Relating Tracks
                %       Now comes the task of relating tracks.  This means updatingthe
                %       matrix that relates the track from one radar with the
                %       track from another radar.
                %       relateTracks.m fuses the necessary vectors.
                %       Tracks that do not require fusion are appended to create
                %       the updateTrks matrix

                [updateTrks,fusedTrks,notFusedTrks]=relateTracks(MostRecentTrks,assocMatrix);

                plotNotFusedTrks = [plotNotFusedTrks;notFusedTrks];
                plotFusedTrks = [plotFusedTrks; fusedTrks];
                % Note that plotNotFusedTrks and plotFusedTrks are kept separately only to simplify
                % to simplify display later and are not required for fusion.

        %       Plot Animation of fusion by plotting every fifth point (15 sec)  for each track
                if PlotMe >=1
                        animationPlots(updateTrks,fusedTrks,notFusedTrks)
                end     % end PlotMe

        %       Updating the Tdbm (Track Database Manager database)
                %       This involves a search for previously reported sites and tracks.
                %       If found, the same Platform Number is assigned. If not a new
                %       Platform Number is generated.The Platform Number is the unique
                %       identifier that will be associated with the Platform Icon
                %       for display to the operator.

                [Tdbm,nextPlatformNo]=TdbmInterface(Tdbm,updateTrks,nextPlatformNo);
                %                       printTdbm(Tdbm);% display of Tdbm if desired.

end             % end of main routine: fusion iteration window


if PlotMe >=1% plot the results at the end.
        outputPlots(plotNotFusedTrks,plotFusedTrks,Tdbm)
end     % end PlotMe
```

timeWindow.m

```
function WindowedObsns = timeWindow(OrigObsnMatrix,timeNow,windowLength)
%	function WindowedObsn = timeWindow(OrigObsnMartix,timeNow,windowLength)
%
%	by:	Major Ian Glenn
%
%
%	Created:		10 Sep 95
%	Modified:		20 Oct 95
%
%
%	Input:
%		OrigObsnMatrix:original obsn matrix to be windowed
%		timeNow:		time (in sec) to window from
%		windowLength:number of seconds to window back in time
%
%	Design:
%		find the rows that meet the time criteria
%		and create the new windowed matrix

rows= find(OrigObsnMatrix(:,7)<=timeNow & OrigObsnMatrix(:,7)>=(timeNow-windowLength));

WindowedObsns=OrigObsnMatrix(rows,:);
```

mostRecentTracks.m

```
function [MostRecentTrks] =  mostRecentTracks(ObsnMatrix)
%
%         function [MostRecentTrks] = mostRecentTracks(ObsnMatrix)
%
%         by:          Major Ian Glenn
%
%
%         Created:              29 Aug 95
%         Modified:             20 Oct 95
%
%         Input:
%          I1                   - ObsnMartix:original obsn matrix to be windowed
%
%
%
%         Output:
%          O1                   - MostRecentTrks
%
%
%
%         Design:
%
%         This function determines which tracks are present and how many
%         in a given set of tracks and plots the distribution
%         and returns the matrix with the most recent observations

MostRecentTrks=[];


[TracksSorted,trackIndx] = sort(ObsnMatrix(:,6));% Sort on field 6 (track)
    % Find the redundancy in a vector x

difference = diff([TracksSorted;max(TracksSorted)+1]);
trackCount = diff(find([1;difference]));
tracksPresent= TracksSorted(find(difference));

% A little graphic feedback
%stem([ tracksPresent],[trackCount]);grid;
%axis([0 max( tracksPresent)+1 0 max(trackCount)+1])



%         It is now easy to select the latest value for each track with
% trackIndx(trackCount(1))

          for i=1:length(tracksPresent)
                    MostRecentTrks=[MostRecentTrks;...
                    ObsnMatrix(trackIndx(sum(trackCount(1:i))),:) ];
          end
```

memshipWSize

```
function [memshipValueOfTrk, thresholdMatrix] = memshipWSize(MostRecentTrks, maxThreshold, minThreshold)

%function [memshipValueOfTrk, thresholdMatrix] = memshipWSize(MostRecentTrks, maxThreshold, minThreshold)
%
%	This function determines the membership of all MostRecentTrkservations as
%	compared to each other MostRecentTrkservation.
%	min & max threshold are not used in this function, but are passed to thresh.m
%	Original code by LT T. Ruthenberg, USN
%
%	by:	Major Ian Glenn
%
%	Written:		5 Sep 95
%	Modified:		23 Oct 95
%
%	Input:
%	  MostRecentTrks - current Observations in the following form:
%			Row 1 - lat [nm]
%			Row 2 - lon [nm]
%			Row 3 - course [nautical degrees]
%			Row 4 - speed [knots]
%			Row 5 - size [metres squared]
%			Row 7 - Site & Track
%			Row 8 - Time [sec]
%			Row 9 - Lost Track
%			Row 10 - Track Quality
%
%      maxThreshold - determines max value of variable threshold function
%      minThreshold - determines min value of variable threshold function
%
%	Output:
%	  memshipValueOfTrk:
%	  thresholdMatrix:
%
%	Design:
%
%	This function determines which tracks are present and how many
%	in a given set of tracks and plots the distribution
%	and returns the matrix with the most recent observations%
%
%	Calls:
%	  thresh.m

MostRecentTrks=MostRecentTrks';% change orientation for algortihm

NMperDEG = 60;
NMperRAD = NMperDEG*180/pi;


[r,col] = size(MostRecentTrks);

% MEMBERSHIP FUNCTION ATTRIBUTES

speed1	=	1;	% accuracy of full membership in speed
size1	=	800;	% accuracy of full membership in size i.e. +/- 400 m^2

memshipValueOfTrk=NaN .* ones(5*(col),col); % create a membership value matrix with NaN
						% assigned to each position representing
						% lat,lon,crse,speed,size

thresholdMatrix=zeros(col,col);

for p=1: col% p refers to the column of the reference track
	refTrk =MostRecentTrks(:,p);
	for n=1:col% Check every other track against the reference track
```

```
if p==n  % do nothing if same track

else

  % Get the threshold to apply for this comparison
  threshold=thresh(refTrk, MostRecentTrks(:,n), maxThreshold, minThreshold);
  thresholdMatrix(p,n)=threshold;% store for posterity


%%%%%%%%%%%%%%%%%%%% POSITION%%%%%%%%%%%%%%

  for m=1:2% X posn is the 1st row, Y posn is in the 2nd row

    if abs((MostRecentTrks(m,n)-refTrk(m,1))) < 500/1852 % 500m coverted to NM

      x=abs((MostRecentTrks(m,n)-refTrk(m,1)));  % calculate the absolute distance
                                                    % between the two positions
      memshipValueOfTrk(5*p-5+m,n) = abs(-x*1852/500 + 1);% Since it falls within the fusion
                                                    % parameters, calculate a
                                                    % Membership value
    else
      memshipValueOfTrk(5*p-5+m,n) = 0;% if outside limits set membership to zero
    end

    if memshipValueOfTrk(5*p-4,n) < threshold

         break                                 % if the membership value assigned does not
                                               % exceed the threshold, STOP, do not even
                                               % look at course, speed and size.

    end

  end

%%%%%%%%%%%%%%%%%% COURSE %%%%%%%%%%%%%
%
%                     course is in the third row and is in degrees


 if memshipValueOfTrk(5*p-4:5*p-3,n)>threshold
  if (MostRecentTrks(3,n)>320 & refTrk(3,1)<40)
        x=360-MostRecentTrks(3,n)+refTrk(3,1);
        memshipValueOfTrk(5*p-2,n) = -x/80 + 1;

  elseif (MostRecentTrks(3,n)<40 & refTrk(3,1)>320)
        x=360-refTrk(3,1)+MostRecentTrks(3,n);
        memshipValueOfTrk(5*p-2,n) = -x/80 + 1;

  elseif abs(MostRecentTrks(3,n) - refTrk(3,1)) < 50
    x=abs(MostRecentTrks(3,n)-refTrk(3,1));
        if x > 180, x=360-x; end
        memshipValueOfTrk(5*p-2,n) = -x/80 + 1;

  else
    memshipValueOfTrk(5*p-2,n) = 0;
  end

 else
  memshipValueOfTrk(5*p-2,n) = NaN;
 end


%%%%%%%%%%%%%%  SPEED %%%%%%%%%%%%%%%%%%%
%
%         speed is in the fourth row and is measured in knots
```

```
    if memshipValueOfTrk(5*p-4:5*p-2,n)>threshold


      if abs(MostRecentTrks(4,n) - refTrk(4,1)) <= speed1

        memshipValueOfTrk(5*p-1,n) = 1;  % if the ref speed falls within +/- 1 knot
                                                  % assign a membership value of 1

      elseif abs(MostRecentTrks(4,n) - refTrk(4,1)) > 1 ...
                      & abs(MostRecentTrks(4,n) - refTrk(4,1)) < 6

        x=abs(MostRecentTrks(4,n)-refTrk(4,1));% It falls within the range
        memshipValueOfTrk(5*p-1,n) = -x/5 + 6/5;% -6 to -1 or 1 to 6 knots
                                                                    % assign appropriate value

      else
        memshipValueOfTrk(5*p-1,n) = 0;
      end

    else
      memshipValueOfTrk(5*p-1,n) = NaN;% if below threshold, fail
    end

  %%%%%%%%%%%%%%%%%%%%  SIZE  %%%%%%
  %
  %        size is in the fifth row and is measured in metres squared


    if memshipValueOfTrk(5*p-4:5*p-1,n)>threshold
                      % check to see all lat,lon,crse&speed passed

      if abs(MostRecentTrks(5,n) - refTrk(5,1)) <= 4000
                                          % if it is within +/- 4000 m^2

        memshipValueOfTrk(5*p,n) = 1;

      elseif abs(MostRecentTrks(5,n) - refTrk(5,1)) > 4000 ...
                      & abs(MostRecentTrks(5,n) - refTrk(5,1)) < 5000
                                                  % between +/- 4000 and +/- 5000 m^2

        x=abs(MostRecentTrks(5,n)-refTrk(5,1));
        memshipValueOfTrk(5*p,n) = -x/1000 + 5;
      else

        memshipValueOfTrk(5*p,n) = 0;
      end

    else

      memshipValueOfTrk(5*p,n) = NaN;% if the other measures have failed
    end

   end % end if p==n

  end % end for n=1:col


end     % end for p=1:col
```

thresh.m

```
function mod_threshold = thresh(ref1, obsn, max_t, min_t)

% function mod_threshold = thresh(ref1, obsn, max_t, min_t)
%       This function determines the threshold based on position of either the reference
%       observation or the observation being compared to.  This function is called by
%       memship.m.  A piecewise linear function is generated from max_t & min_t
%       Original code by LT T. Ruthenberg, USN
%
%       by:     Major Ian Glenn
%
%       Modified:       28 Oct 95
%
%       INPUTS:
%       I1      ref1    - the reference observation
%       I2      obsn    - the observation being compared to the reference
%       I3      max_t   - the maximum threshold
%       I4      min_t   - the minimum threshold
%
%       OUTPUTS:
%       O1      mod_threshold   - the threshold modified based on distance
%                                       from the radar.
%
%%%%%%%%% Set Radar Locations %%%%%%%%%%%%

rdr1x = -01- 05.53583/60;       %displacement of radar 1 in lat & lon
rdr1y = 41+18.59502/60;         % Governors Island

rdr2x = -05 - 25.33414/60;      %displacement of radar 2 Bank Street
rdr2y = 38+48.5168/60 ;

rad_loc = [          rdr1x,    rdr2x;...
                     rdr1y,    rdr2y];

NMperDEG = 60;
NMperRAD = NMperDEG*180/pi;

%       site or radar producing the track is contained in the sixth row
refRdrXPosn = rad_loc(1,floor(ref1(6)/1000));           % select the radar that produced the track
refRdrYPosn = rad_loc(2,floor(ref1(6)/1000));
refXPosn = ref1(1) ;
refYPosn = ref1(2);

%       Distance from the observing radar to the referenced observation in NM
dist_ref=sqrt(((refRdrXPosn-refXPosn))^2 + ((refRdrYPosn-refYPosn))^2 );

obsnRdrXPosn = rad_loc(1,floor(obsn(6)/1000));          % select the radar that produced the track
obsnRdrYPosn = rad_loc(2,floor(obsn(6)/1000));
obsnXPosn = obsn(1) ;
obsnYPosn = obsn(2);

%       Distance from the observing radar to the observation being compared
dist_obs=sqrt(((obsnRdrXPosn-obsnXPosn))^2 + ((obsnRdrYPosn-obsnYPosn))^2 );

if dist_ref<=1 & dist_obs<=1
        mod_threshold=min_t;
elseif (dist_ref > 1 | dist_obs > 1) & (dist_ref < 5 & dist_obs < 5)
        mod_threshold=((max_t-min_t)/4)*min([dist_ref dist_obs]) + min_t-((max_t-min_t)/4);
else
        mod_threshold=max_t;
end

if mod_threshold < min_t
        mod_threshold=min_t;
end
```

associationMatrix.m

```
function assocMatrix = associationMatrix(memshipValueOfTrk, thresholdMatrix)

%       function assocMatrix = associationMatrix(memshipValueOfTrk, thresholdMatrix)
%       This function generates the association matrix based on the current
%       membership values and the minimum threshold provided by thresholdMatrix
%       Original code by LT T. Ruthenberg, USN
%
%       by:         Major Ian Glenn
%
%       Modified:           23 Oct 95
%
%       INPUTS:
%       I1          memshipValueOfTrk         - membership values of current observations
%       I2          thresholdMatrix           - minimum allowable threshold for observations
%
%       OUTPUTS:
%       O1          assocMatrix               - association matrix
%

[r,col]=size(memshipValueOfTrk);
assocMatrix=zeros(col,col);
nn=0;

for p=1:5:r-4
  nn=nn+1;
  for n=nn:col
%                   disp('assocMatrix values')

                    threshold=thresholdMatrix(n,ceil(p/5));
                    if memshipValueOfTrk(p:p+4,n) > threshold
                              assocMatrix(ceil(p/5),n)=1;
                    end
  end
end
```

relateTracks.m

```
function [updateTrks,fusedTrks,notFusedTrks]=relateTracks(MostRecentTrks,assocMatrix)

%       This function creates carries out the actual association of tracks
%
%       Written by:  Major Ian Glenn
%
%       Created:            13 Oct 95
%       Modified:           19 Oct 95
%
%       Input:
%       I1      mostRecentTrks    - these are the tracks to associate or not
%       I2      assocMatrix       - dictates which tracks should be associated
%
%
%       Output:
%       O1      relatedTracks     - final matrix of tracks to update with Tdbm
%                                         as unique platform tracks
%       O2      fusedTrks         - use this to plot fused tracks
%
%       Design:
%               Use assocMatrix to index which tracks to fuse
%
%


assocMatrix=assocMatrix-diag(diag(assocMatrix));% this step subtracts out the main
                                                 % diagonal elements (relating self to self)

[row,colm]=find(assocMatrix);% find the element locations where relation appears
                             % the rows give the first track to associate
                             % and the colms the second track
%if length(row)>0
%       disp(sprintf(' Fuse tracks %4d and %4d \n', MostRecentTrks(row,6),MostRecentTrks(colm,6)));
%end

sitesAndTrks = MostRecentTrks(row,6)*10000+MostRecentTrks(colm,6);

% Now comes the time to fuse the tracks using the centroid weighted by track quality

fusedLat = ((MostRecentTrks(row,1)   .*MostRecentTrks(row,9)./9) ...
                        +(MostRecentTrks(colm,1)  .*MostRecentTrks(colm,9)./ 9 ))./ 2 ;

fusedLon = ((MostRecentTrks(row,2)   .*MostRecentTrks(row,9)./9) ...
                        +(MostRecentTrks(colm,2)  .*MostRecentTrks(colm,9)./ 9 ))./ 2 ;

fusedCrse = ((MostRecentTrks(row,3)  .*MostRecentTrks(row,9)./9) ...
                        +(MostRecentTrks(colm,3)  .*MostRecentTrks(colm,9)./ 9 ))./ 2 ;

fusedSpeed = ((MostRecentTrks(row,4)  .*MostRecentTrks(row,9)./9) ...
                        +(MostRecentTrks(colm,4)  .*MostRecentTrks(colm,9)./ 9 ))./ 2 ;

fusedSize = ((MostRecentTrks(row,5)  .*MostRecentTrks(row,9)./9) ...
                        +(MostRecentTrks(colm,5)  .*MostRecentTrks(colm,9)./ 9 ))./ 2 ;

fusedTime = max([MostRecentTrks(row,7),MostRecentTrks(colm,7)]')';

fusedLT = MostRecentTrks(row,8)+MostRecentTrks(colm,8);

fusedTQ = MostRecentTrks(row,9)*10+MostRecentTrks(colm,9);

fusedTrks = [fusedLat,fusedLon,fusedCrse,fusedSpeed,fusedSize,sitesAndTrks,...
                              fusedTime,fusedLT,fusedTQ];
```

```
% now append the tracks that are not to be fused
mask=ones(size(assocMatrix))-assocMatrix;
rowTotal=sum(mask);
colmTotal=sum(mask');
notFusedTrksIndx= find(rowTotal==colmTotal);
notFusedTrks = MostRecentTrks(notFusedTrksIndx,:);


%disp('Not Fused Tracks')
%printPretty(MostRecentTrks(notFusedTrksIndx,:))

updateTrks = [fusedTrks;notFusedTrks];

%disp('Tracks to Update Tdbm')
%printPretty(updateTrks)
```

TdbmInterface.m

```
function [Tdbm,nextPlatformNo]=TdbmInterface(Tdbm,updateTrks,nextPlatformNo)

%       This function simulates the interface to the Tdbm.
%
%       Written by:  Major Ian Glenn
%
%       Created:            19 Oct 95
%       Modified:           20 Oct 95
%
%       Input:
%       I1                  - Tdbm              The Track database
%       I2                  - updateTrks
%       I3                  - PlatformNo        A unique platform number
%
%       Output:
%       O1                  - Tdbm
%       O2                  - PlatformNo        this is the global unique track no.
%
%
%       Design:
%               For each track in updateTrks
%                       Search the Tdbm for a matching site&track number
%                       if match
%                               copy PlatformNo to track
%
%                       if no match
%                               create a new PlatformNo
%                               append to track
%                               update PlatformNo counter
%               Append new tracks with their PlatformNos assigned to Tdbm
%

% Initialize structure

if exist('Tdbm') ~= 1           % if this is the first time the Tdbm is called
        initPlatformNo = (1:size(updateTrks,1))';
        Tdbm = [updateTrks,initPlatformNo];
        nextPlatformNo = size(updateTrks,1) +1;
        return
end

% Get the site&tracks stored in the Tdbm

TdbmSiteNtrks = [floor(Tdbm(:,6)/10000),Tdbm(:,6)-floor(Tdbm(:,6)/10000)*10000];

% now check them against the sites&tracks in the new info

for rowUpdatedTrk=1:size(updateTrks,1)% look at every row in updateTrks matrix

        % extract the site&track numbers in a vector like [1001 2001]
        UpdateTrkSiteNtrk =[floor(updateTrks(rowUpdatedTrk,6)/10000),...
                                        updateTrks(rowUpdatedTrk,6)-floor(updateTrks(rowUpdat-
                                edTrk,6)/10000)*10000];
%-------------------------------------------------------------------------------------------------
        noSiteNTrksPresent=length(find(UpdateTrkSiteNtrk));

        if noSiteNTrksPresent ==1% when only one site&track is present

                [rowPlatformNo, notUsed] =find(TdbmSiteNtrks == UpdateTrkSiteNtrk(2));% Get the site&track
                                number

                if length(rowPlatformNo) ==0  % if no match is found
                        PlatformNo(rowUpdatedTrk) = nextPlatformNo;% assign the next available Platform
                                number
```

```
                                        % to the vector of Platform numbers
                        nextPlatformNo = nextPlatformNo +1;% increment the Platform number

                else                                                        % if a match is found
                        sameTrackRow = max(rowPlatformNo);          % grab the latest matching track in
                                        Tdbm
                        PlatformNo(rowUpdatedTrk) = Tdbm(sameTrackRow,10) ; % and grab out the Platform
                                        No
                end

        elseif noSiteNTrksPresent ==2           % when two site&tracks are present

                [rowPlatformNo, notUsed] =find(TdbmSiteNtrks == UpdateTrkSiteNtrk(1));% Get the 1st
                                        site&track number

                if length(rowPlatformNo) ==0  % if no match is found on first site&track,
                                                                            %        check
                                        second site&track

                        [rowPlatformNo, notUsed] =find(TdbmSiteNtrks == UpdateTrkSiteNtrk(2));

                                        % Get the site&track number for 2nd site&track


                        if length(rowPlatformNo) ==0  % if no match is found on second site&track

                                PlatformNo(rowUpdatedTrk) = nextPlatformNo;% assign the next available
                                        Platform number

                                        % to the vector of Platform numbers
                                nextPlatformNo = nextPlatformNo +1;% increment the Platform number

                        else                                                % if a match is
                                        found on 2nd site&track

                                sameTrackRow = max(rowPlatformNo);          % grab the latest matching
                                        track in Tdbm
                                PlatformNo(rowUpdatedTrk) = Tdbm(sameTrackRow,10) ; % and grab out the
                                        Platform No

                        end


                else    % if a match is found on the first site&track

                        sameTrackRow = max(rowPlatformNo);% grab the latest matching track in Tdbm
                        PlatformNo(rowUpdatedTrk) = Tdbm(sameTrackRow,10) ; % and grab out the Platform
                                        No

                end

        end

end                     % end for loop to check each row of updateTrks

PlatformNo = PlatformNo';

% Append the new tracks to the Tdbm

Tdbm = [Tdbm ; updateTrks , PlatformNo];
return
```

randomSize.m

```
%function ObsnMatrixRandSize = randomSize()
%
%       Written by:  Major Ian Glenn
%
%       Created:            25 Oct 95
%       Modified:           25 Oct 95
%
%       Input:
%       I1                              -Simulation data
%
%       Output:
%       O1                              - ObsnMatrixRandSize with size randomized as follows:
%       O2                              - size - 1 std dev to + 2 std dev
%                           mean     Std Deviation              Min             Max
%       Ship A              800           200                   600             1200
%       Ship B              1200          500                   700             2200
%       Ship C              1600          600                   1000            2800
%       Ship D              3200          1200                  2000            5600
%
%       Design:
%                           load the simulation data and randomize the size value
%

        if exist('TrackTableRdr1') == 0
                load TrackTableGIa.mat;
        end

        if exist('TrackTableRdr2') == 0
                load TrackTableBanka.mat;

        end

        if exist('TrackTableRdr1b') == 0
                load TrackTableGIb.mat;
        end
        if exist('TrackTableRdr2b') == 0
                load TrackTableBankb.mat;
        end

        ObsnMatrixRandSize = [TrackTableRdr1;TrackTableRdr2;TrackTableRdr1b;TrackTableRdr2b];


x = sort(ObsnMatrixRandSize(:,5));
difference=diff([x;NaN]);
sizesPresent = x(difference~=0)

%       Ship A              800             200                 600     1200


resultFind800 = find(ObsnMatrixRandSize(:,5) == 800);
tempA = randn(length(resultFind800),1)*200 + 800;
lowSide = 600
highSide = 1200

% Limit to one std dev below

tempA(find(tempA < lowSide)) = ones(length(find(tempA < lowSide)),1).*lowSide;

% Limit to two std dev above

tempA(find(tempA > highSide)) = ones(length(find(tempA >highSide)),1).*highSide;

figure(1)
subplot(221);hist(tempA)
```

```
title('Ship A')
xlabel('meters')
ylabel('count')



%       Ship B     1200          500           700      2200

resultFind1200 = find(ObsnMatrixRandSize(:,5) == 1200);
tempB = randn(length(resultFind1200),1)*500 + 1200;
lowSide = 700
highSide = 2200
tempB(find(tempB < lowSide)) = ones(length(find(tempB < lowSide)),1).*lowSide;

% Limit to two std dev above

tempB(find(tempB > highSide)) = ones(length(find(tempB >highSide)),1).*highSide;
subplot(222);hist(tempB)
title('Ship B')
xlabel('meters')
ylabel('count')


%       Ship C     1600          600           1000     2800


resultFind1600 = find(ObsnMatrixRandSize(:,5) == 1600);
tempC = randn(length(resultFind1600),1)*600 + 1600;
lowSide = 1000
highSide = 2800
tempC(find(tempC < lowSide)) = ones(length(find(tempC < lowSide)),1).*lowSide;

% Limit to two std dev above

tempC(find(tempC > highSide)) = ones(length(find(tempC > highSide)),1).*highSide;
subplot(223);hist(tempC)
title('Ship C')
xlabel('meters')
ylabel('count')


%       Ship D     3200          1200          2000     5600

resultFind3200 = find(ObsnMatrixRandSize(:,5) == 3200);
tempD = randn(length(resultFind3200),1)*1200 + 3200;
lowSide = 2000
highSide = 5600
tempD(find(tempD < lowSide)) = ones(length(find(tempD < lowSide)),1).*lowSide;

% Limit to two std dev above

tempD(find(tempD > highSide)) = ones(length(find(tempD >highSide)),1).*highSide;
subplot(224);hist(tempD)
title('Ship D')
xlabel('meters')
ylabel('count')

ObsnMatrixRandSize(resultFind800,5)= tempA;

ObsnMatrixRandSize(resultFind1200,5)= tempB;

ObsnMatrixRandSize(resultFind1600,5)= tempC;

ObsnMatrixRandSize(resultFind3200,5)= tempD;
```

```
 print figRandSizeDistrs

figure(2)
hist(ObsnMatrixRandSize(:,5))
title('Overall Size Distribution')
xlabel('meters')
ylabel('count')

 print figOverallSizeDistr
```

showChart1.m

```
%function []=showChart1()

% function
%
% This function creates the underlying harbor chart graphics used for plotting results
%
%         Written by:  Major Ian Glenn
%
%         Created:              14 Oct 95
%         Modified:             15 Oct 95
%
if exist('Y') == 0
          [X,map]=gifread('NYHbr.gif') ;

          Y= flipud(X);
end
clf

chartHandle= image(Y)  ;
colormap(map)
axis('xy')
%axis('image')
 axis('square')

% change aspect ratio of chart so that lat and lon are equal
% current is 1.37 width to 1.8 height
% therefore multiply width of 818 pixels by 1.8/1.37

set(chartHandle,'XData',[1 818*1.8/1.37]) % now onw unit equals one unit in each dim
 axis([50 , 1000, 250, 950])
% set(chartHandle,'Visible','off')
% set(chartHandle,'Visible','on')

fig = gcf  ;             % Get current figure handle.
imageHandle = get(fig,'Children');

set(imageHandle,'XColor','b')
set(imageHandle,'YColor','b')

 set(imageHandle,'GridLineStyle','-')
set(imageHandle,'XGrid','on')
set(imageHandle,'YGrid','on')
 set(imageHandle,'XTick',[130, 825])
 set(imageHandle,'YTick',[635, 810])
 set(imageHandle,'XTickLabels',['74,05W';'74,01W'])
set(imageHandle,'YTickLabels',['40,40N';'40,41N'])

%imzoom
hold on

 plotHandle= axes('position',[0.13, 0.11, 0.775, 0.815]) ;  %[left, bottom, width, height]
%set(H,'Color','none')
set(plotHandle,'XColor','g')
set(plotHandle,'YColor','g')
set(plotHandle,'Box','on')
 axis('square')
grid
 axis([-5.46 -0 37.8 41.8])

 set(plotHandle,'GridLineStyle','-.')
set(plotHandle,'XGrid','on')
set(plotHandle,'YGrid','on')
 set(plotHandle,'XTick',[-1, -2,-3,-4,-5])
 set(plotHandle,'YTick',[38, 39,40,41])
 set(plotHandle,'XTickLabels',['01W';'02W';'03W';'04W';'05W'])
set(plotHandle,'YTickLabels',['38N';'39N';'40N';'41N'])
 set(plotHandle,'NextPlot','add')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 set(chartHandle,'Visible','on')
 set(imageHandle,'Visible','off')
 set(plotHandle,'Visible','on')

 % get(plotHandle)

GIlat=-01 - 05.53583/60;
GIlon=41+18.59502/60 ;
 plot(GIlat,GIlon,'b*')
rdr1x=GIlat;
rdr1y=GIlon;


Banklat=-05 - 25.33414/60;
Banklon=38+48.5168/60 ;
 plot(Banklat,Banklon,'r*')
rdr2x=Banklat;
rdr2y=Banklon;

h3=line([GIlat GIlat],[GIlon-.15,GIlon+.15]);
          set(h3,'Linestyle','-')
          set(h3,'color','b')

set(0,'DefaultTextColor','blue')  %This makes the radar sight print in blue
set(0,'DefaultTextFontSize',14)

          text(rdr1x+.1,rdr1y-.3, 'Radar 1:  ')
          text(rdr1x+.1,rdr1y-.5, 'Governor''s ')
          text(rdr1x+.1,rdr1y-.7, 'Island    ')

h4=line([rdr1x-.15 rdr1x+.15],[rdr1y,rdr1y]);
          set(h4,'Linestyle','-')
          set(h4,'color','b')

h5=line([rdr2x rdr2x],[rdr2y-.15,rdr2y+.15]);
          set(h5,'Linestyle','-')
          set(h5,'color','r')
set(0,'DefaultTextColor','red')  %This makes the radar sight print in blue
          text(rdr2x+.05,rdr2y-.3,'Radar 2:')
          text(rdr2x+.05,rdr2y-.5,'Bank St')

h6=line([rdr2x-.15 rdr2x+.15],[rdr2y,rdr2y]);
          set(h6,'Linestyle','-')
          set(h6,'color','r')

set(0,'DefaultTextFontSize',9)
set(0,'DefaultTextColor','white')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 trk1Handle= axes('position',[0.13, 0.11, 0.775, 0.815]) ; %[left, bottom, width, height]
%set(H,'Color','none')
set(trk1Handle,'XColor','m')
set(trk1Handle,'YColor','m')
set(trk1Handle,'Box','on')
 axis('square')
grid
 axis([-5.46 -0 37.8 41.8])

 set(trk1Handle,'GridLineStyle','-.')
set(trk1Handle,'XGrid','on')
set(trk1Handle,'YGrid','on')
 set(trk1Handle,'XTick',[-1, -2,-3,-4,-5])
 set(trk1Handle,'YTick',[38, 39,40,41])
 set(trk1Handle,'XTickLabels',['01W';'02W';'03W';'04W';'05W'])
set(trk1Handle,'YTickLabels',['38N';'39N';'40N';'41N'])
 set(trk1Handle,'NextPlot','add')
% set(trk1Handle,'FontSize',12)
 set(trk1Handle,'Visible','off')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Create the box to show coverage areas
xf1=[-4,-0,-0,-4];
 yf1=[41.8,41.8,39,39];

gf1=fill(xf1,yf1,'c');
 set(gf1,'EdgeColor','c');
set(gf1,'LineWidth',3)
set(gf1,'FaceColor','none')
 %set(g1,'Visible','off')


% Create the box to show coverage areas
xf2=[-5.46,-02,-02,-5.46];
 yf2=[40.5,40.5,37.8,37.8];

gf2=fill(xf2,yf2,'m');
 set(gf2,'EdgeColor','m');
set(gf2,'LineWidth',3)
set(gf2,'FaceColor','none')
 %set(g2,'Visible','off')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% use cla to clear a track from the axis
```

inputPlots.m

```
function inputPlots(TrackTableRdr1,TrackTableRdr2,TrackTableRdr1b,...
                        TrackTableRdr2b, ObsnMatrix)
%
%       Written by:  Major Ian Glenn
%
%       Created:            20 Oct 95
%       Modified:           22 Nov 95
%
%       Input:    Track data from FusionAlgorithm.m
%       I1                          - TrackTableRdr1
%       I2                          - TrackTableRdr2
%       I3                          - TrackTableRdr1b
%       I4                          - TrackTableRdr2b
%       I5                          - ObsnMatrix
%
%       Output:
%               Plots of input data
%
%       Design:
%               plot routines to show inputs to the algorithm
%
%       Calls:
%       showChart1.m
%       showChart3.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       ****************************** Plot Fcns *********************
                figure(1); % track 1001
                clg
                showChart1

                set(0,'DefaultTextFontName','times-bold')
                set(0,'DefaultTextFontSize',18)
                plot(TrackTableRdr1(:,1),TrackTableRdr1(:,2),'bo')
                set(0,'DefaultTextColor', 'b')
                text(TrackTableRdr1(100,1)+.1,TrackTableRdr1(100,2),...
                                num2str(TrackTableRdr1(1,6)))

print -dpsc figtrk1001
print -dgif8 figtrk1001


                figure(2); % track 2001 & 2002
                clg
                showChart1

                set(0,'DefaultTextFontName','times-bold')
                set(0,'DefaultTextFontSize',18)
                plot(TrackTableRdr2(:,1),TrackTableRdr2(:,2),'r+')
                set(0,'DefaultTextColor', 'r')
                text(TrackTableRdr2(30,1)+.1,TrackTableRdr2(30,2), num2str(TrackTableRdr2(30,6)))
                text(TrackTableRdr2(530,1)+.1,TrackTableRdr2(530,2), num2str(TrackTableRdr2(500,6)))

                figure(3); % 1003 & 1004
                clg
                showChart1

                set(0,'DefaultTextFontName','times-bold')
                set(0,'DefaultTextFontSize',18)
                plot(TrackTableRdr1b(:,1),TrackTableRdr1b(:,2),'bo')
                set(0,'DefaultTextColor', 'b')
                text(TrackTableRdr1b(30,1)+.1,TrackTableRdr1b(30,2), num2str(TrackTableRdr1b(30,6)))
                text(TrackTableRdr1b(750,1)+.1,TrackTableRdr1b(750,2), num2str(TrackTableRdr1b(750,6)))
```

```
                    figure(4); % 2003 & 2004
                    clg
                    showChart1
                    set(0,'DefaultTextFontName','times-bold')
                    set(0,'DefaultTextFontSize',18)
                    plot(TrackTableRdr2b(:,1),TrackTableRdr2b(:,2),'r+')
                    set(0,'DefaultTextColor', 'r')
                    text(TrackTableRdr2b(10,1)+.1,TrackTableRdr2b(10,2), num2str(TrackTableRdr2b(10,6)))
                    text(TrackTableRdr2b(750,1)+.1,TrackTableRdr2b(750,2), num2str(TrackTableRdr2b(750,6)))

                    figure(5); % combined plot
                    clg
                    showChart1
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==1001),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==1001),2),'r*')
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==1002),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==1002),2),'r*')
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==1003),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==1003),2),'r*')
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==1004),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==1004),2),'r*')
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==2001),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==2001),2),'b+')%
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==2002),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==2002),2),'b+')%
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==2003),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==2003),2),'b+')
                    plot(ObsnMatrix(find(ObsnMatrix(:,6)==2004),1),ObsnMatrix(find(ObsnMa-
                                        trix(:,6)==2004),2),'b+')

print -dpsc figInputTrks
print -dgif8 figInputTrks
```

animationPlots.m

```
function animationPlots(updateTrks,fusedTrks,notFusedTrks)
%
%        Written by:  Major Ian Glenn
%
%        Created:            20 Oct 95
%        Modified:           22 Nov 95
%
%        Input:    Track data from FusionAlgorithm.m
%        I1                              - updateTrks
%        I2                              - fusedTrks
%        I3                              - notFusedTrks
%
%        Output:
%
%
%        Design:
%                            plot routines to show progress of the algorithm
%
%        Calls:
%        showChart1.m
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%        ***************************** Plot Fcns ********************
figure(6)
% Put on the site and track numbers every 50th iteration.
if floor(updateTrks(1,7)/50) ~= floor(((updateTrks(1,7)-1)/50))
        showChart1
        set(0,'DefaultTextColor', 'b')
                set(0,'DefaultTextFontName','times-bold')
                set(0,'DefaultTextFontSize',18)

        for g= 1:size(updateTrks,1)
                text(updateTrks(g,1)+.1,updateTrks(g,2), sprintf('%d',updateTrks(g,6)))
        end
        set(0,'DefaultTextFontSize',9)
end     % end floor

%       Plot every fifth point.
if floor(updateTrks(1,7)/5) ~= floor(((updateTrks(1,7)-1)/5))
        % print ever 5 pts or 15 seconds (3 second updates)
        if length(notFusedTrks ) >0
                plot(notFusedTrks(:,1),notFusedTrks(:,2),'b*')
                plot(notFusedTrks(:,1),notFusedTrks(:,2),'b.')
        end
        if length(fusedTrks) >0
                plot(fusedTrks(:,1),fusedTrks(:,2),'ro')
                plot(fusedTrks(:,1),fusedTrks(:,2),'r.')
        end

        set(0,'DefaultTextColor', 'b')

        % Create the box to show time
        xf1=[-1.9,-0.1,-0.1,-1.9];
         yf1=[38,38,38.4,38.4];

        gf1=fill(xf1,yf1,'b');
        set(gf1,'EdgeColor','b');
        set(gf1,'LineWidth',3)
        set(gf1,'FaceColor','y')
        set(0,'DefaultTextFontName','times-bold')
        set(0,'DefaultTextFontSize',12)
        handle1 = text(-1.8,38.2, ['Time: ' , num2str(updateTrks(1,7)),' sec']);
```

```
% Print out some interesting time epochs
if updateTrks(1,7) == 165
            print -dpsc figTime165
            print -dgif8 figTime165
end

if updateTrks(1,7) == 585
            print -dpsc figTime585
            print -dgif8 figTime585
end

if updateTrks(1,7) == 870
            print -dpsc figTime870
            print -dgif8 figTime870
end

if updateTrks(1,7) == 975
            print -dpsc figTime975
            print -dgif8 figTime975
end

if updateTrks(1,7) == 1125
            print -dpsc figTime1125
            print -dgif8 figTime1125
end

if updateTrks(1,7) == 1335
            print -dpsc figTime1335
            print -dgif8 figTime1335
end

            set(0,'DefaultTextFontSize',9)

end         % end plot every 10 pts

            set(0,'DefaultTextColor', 'default')
```

outputPlots.m

```
function  outputPlots(plotNotFusedTrks,plotFusedTrks,Tdbm)
%
%
%        Written by:  Major Ian Glenn
%
%        Created:              20 Oct 95
%        Modified:             22 Nov 95
%
%        Input:    Track data from FusionAlgorithm.m
%        I1                             - plotNotFusedTrks
%        I2                             - plotFusedTrks
%        I3                             - Tdbm
%
%        Output:
%        O1                             -   figOutputPlatforms.ps
%
%        Design:
%                             plot routines
%
%        Calls:
%        showChart1.m
%        showChart2.m
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         figure(7)  % plot of fused and not fused tracks
         clg
         showChart1

         plot(plotNotFusedTrks(:,1),plotNotFusedTrks(:,2),'c+')
         plot(plotNotFusedTrks(:,1),plotNotFusedTrks(:,2),'b.')

         plot(plotFusedTrks(:,1),plotFusedTrks(:,2),'mo')
         plot(plotFusedTrks(:,1),plotFusedTrks(:,2),'k.')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         figure(8)  % zoomed view of fused and not fused tracks
         clg
         showChart2


         plot(plotNotFusedTrks(:,1),plotNotFusedTrks(:,2),'c+')
         plot(plotNotFusedTrks(:,1),plotNotFusedTrks(:,2),'b.')


         plot(plotFusedTrks(:,1),plotFusedTrks(:,2),'mo')
         plot(plotFusedTrks(:,1),plotFusedTrks(:,2),'k.')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
         figure(9)  % plot of resultant platform tracks in Tdbm
         clg
         showChart1


set(0,'DefaultTextFontSize',12)
plot(Tdbm(find(Tdbm(:,10)==1),1),Tdbm(find(Tdbm(:,10)==1),2),'b+')
set(0,'DefaultTextColor', 'b')
platform1= [Tdbm(find(Tdbm(:,10)==1),1),Tdbm(find(Tdbm(:,10)==1),2)];
text(platform1(30,1)+.2,platform1(30,2), 'Platform 1')

plot(Tdbm(find(Tdbm(:,10)==2),1),Tdbm(find(Tdbm(:,10)==2),2),'ro')
set(0,'DefaultTextColor', 'r')
platform2= [Tdbm(find(Tdbm(:,10)==2),1),Tdbm(find(Tdbm(:,10)==2),2)];
text(platform2(30,1)+.2,platform2(30,2), 'Platform 2')
```

```
plot(Tdbm(find(Tdbm(:,10)==3),1),Tdbm(find(Tdbm(:,10)==3),2),'g*')
set(0,'DefaultTextColor', 'g')
platform3= [Tdbm(find(Tdbm(:,10)==3),1),Tdbm(find(Tdbm(:,10)==3),2)];
text(platform3(220,1)+.1,platform3(220,2), 'Platform 3')

plot(Tdbm(find(Tdbm(:,10)==4),1),Tdbm(find(Tdbm(:,10)==4),2),'mx')
set(0,'DefaultTextColor', 'm')
platform4= [Tdbm(find(Tdbm(:,10)==4),1),Tdbm(find(Tdbm(:,10)==4),2)];
text(platform4(270,1)+.1,platform4(270,2), 'Platform 4')

print -dpsc figOutputPlatforms
print -dgif8 figOutputPlatforms
```

plotMembership.m

```
%function [memshipValueOfTrk, thresholdMatrix] = plotMembership(MostRecentTrks, maxThreshold, minThreshold)
%
%
%       This function ot the memebership functions used in the algorithm
%
%
%       Written by: Major Ian Glenn
%
%       Created:        28 Oct 95
%       Modified:       23 Oct 95
%


% MEMBERSHIP FUNCTION ATTRIBUTES

speed1  =       1;      % accuracy of full membership in speed
size1   =       800;    % accuracy of full membership in size i.e. +/- 400 m^2

if 1
%%%%%%%%%%%%%%% POSITION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xValue = linspace(-.7,.7,200) ; % -.7 to .7 NM
yValue = zeros(size(xValue));
for ctr=1: length(xValue)
            if abs(xValue(ctr)) < 500/1852 % 500m coverted to NM

              yValue(ctr) = abs(-abs(xValue(ctr))*1852/500 + 1);
            else
              yValue(ctr) = 0;% if outside limits set membership to zero
            end

end
subplot(221)
plot(xValue*1852,yValue,'b')
axis([-1000 1000 0 1.2])
xlabel('meters');ylabel('membership')
title('Lat and Lon')



        %%%%%%%%%%%%%  COURSE %%%%%%%%%%%%%%%%%%%%%%%%5
        %
        %               course is in the third row and is in degrees
refTrkCrse = 90
MostRecentTrksCrse = linspace(0,360,200) ; % perform relative degree check
xValue = linspace(-30,30) ; % -30 to 30 degrees
yValue = zeros(size(xValue));

for ctr=1: length(MostRecentTrksCrse)


          if (MostRecentTrksCrse(ctr)>320 & refTrkCrse<40)
%                 disp('>320 & <40')
                        x=360-MostRecentTrksCrse(ctr)+refTrkCrse
                        yValue(ctr) = -x/80 + 1;

          elseif (MostRecentTrksCrse(ctr)<40 & refTrkCrse>320)
%                 disp('<40 & >320')
                        x=360-refTrkCrse+MostRecentTrksCrse(ctr)
                        yValue(ctr) = -x/80 + 1;

          elseif abs(MostRecentTrksCrse(ctr) - refTrkCrse) < 50
%                 disp('diff<50')

            x=abs(MostRecentTrksCrse(ctr)-refTrkCrse);
```

```
                        if x > 180, x=360-x; end
%                  disp('x > 180, x=360-x')

                                yValue(ctr) = -x/80 + 1;

                   else
                      yValue(ctr) = 0;
                   end

                 end
subplot(222)
plot(MostRecentTrksCrse-refTrkCrse,yValue,'b')
axis([-90 90 0 1.2])
xlabel('degrees');ylabel('membership')
title('Crse')

        %%%%%%%%%%%%%%%%  SPEED  %%%%%%%%%%%%%%%%%%
        %
        %          speed is in the fourth row and is measured in knots
relativeSpeed = linspace(-8,8,200) ; % perform relative SPEED check

memshipValueOfTrk = zeros(size(relativeSpeed));

for ctr=1: length(relativeSpeed)


            if abs(relativeSpeed(ctr)) <= speed1

              memshipValueOfTrk(ctr) = 1 ;
                                                % if the ref speed falls within +/- 1 knot
                                               % assign a membership value of 1

            elseif abs(relativeSpeed(ctr)) > 1 & abs(relativeSpeed(ctr)) < 6

              x=abs(relativeSpeed(ctr));% It falls within the range
              memshipValueOfTrk(ctr) = -x/5 + 6/5;
                                                                    % -6 to -1 or 1 to 6 knots
                                                                    % assign appropriate value

            else
              memshipValueOfTrk(ctr) = 0;
            end


end % end ctr
           subplot(223)
plot(relativeSpeed,memshipValueOfTrk,'b')
axis([-8 8 0 1.2])
xlabel('knots');ylabel('membership')
title('Speed')
end % if 0

        %%%%%%%%%%%%%%%%%%%%  SIZE  %%%%%%
        %
        %          size is in the fifth row and is measured in metres squared

relativeSize = linspace(-5000,5000,200) ; % perform relative SPEED check

memshipValueOfTrk = zeros(size(relativeSize));

for ctr=1: length(relativeSize)



            if abs(relativeSize(ctr)) <= 2000
                                    % if it is within +/- 2000 m^2
```

```
      memshipValueOfTrk(ctr) = 1;

    elseif abs(relativeSize(ctr)) > 2000 ...
                    & abs(relativeSize(ctr)) < 3000
                                                        % between +/- 2000 and +/- 3000 m^2

      x=abs(relativeSize(ctr));
      memshipValueOfTrk(ctr) = -x/1000 + 3;
    else

      memshipValueOfTrk(ctr) = 0;
    end



end % end ctr

        subplot(224)
plot(relativeSize,memshipValueOfTrk,'b')
axis([-5000 5000 0 1.2])
xlabel('square meters');ylabel('membership')
title('Size')
```

APPENDIX B. SIMULATION CODE

This appendix contains the code used to perform the Kalman filtering of the simulated tracks. KalmanTrackerCD was used to filter the data sets for ships C and D, the identical KalmanTrackerAB (not listed) was used to filter ships A and B. The outputs were used to construct a simulated database of link tracks for the algorithm in Appendix A to process. The following code is contained in this appendix:

- **KalmanTrackerCD.m** on page 82.
- **PseudoMeasure.m** on page 93.
- **KalmanPredict.m** on page 95.
- **KalmanUpdate.m** on page 96.
- **Init_PosnVel** on page 97.

KalmanTrackerCD.m

```
%       KalmanTrackerCD.m
%
%       MULTI-SENSOR, MULTI-TARGET TRACKING
%       OF SHIPPING TRAFFIC IN NEW YORK HARBOUR
%       BY THE VESSEL TRAFFIC SYSTEM.
%
%
%       Written by:     Major Ian Glenn
%
%       Created:        14 Oct 95
%       Modified:       18 Oct 95
%
%
%
%       Associated files:
%       Input:                  Init_PosnVel.m- Initialization routine
%                               PseudoMeasure.m - Take data and make
%                                                 into x&y
%                               KalmanPredict.m
%                               KalmanUpdate.m
%                               ShipC.mat       - Simulink track data
%                               ShipD.mat       - Simulink track data
%
%       Output:                 TrackTableGIb.mat       - TrackTableRdr1
%                               TrackTableBankb.mat     - TrackTableRdr2
%
%       Problem Statement:
%       Using the manoeuvre simulation to generate the actual
%       target motion, use a filter to track the target in
%       the xy plane
%       Tranform coordinates for radar sites
%       Build the Track Table
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Initialize variables
%
%       Assume identical radar specs
crse1=[NaN ;NaN];speed1=[NaN ;NaN];crse2=[NaN ;NaN];speed2=[NaN ;NaN];
%*****************************************************************************
Radar = 2           % 1== Governor's Island and 2 == Bank Street
PrintMe = 1         % 1 == print figures
PlotMe = 2          % 1= see final plot   2 = see all plots
%*****************************************************************************

Site =['Governor''s Island';
   'Bank Street      '];
disp('Using the following Radar Site:')
disp(Site(Radar,:))

delta_t = 3;            % Real update times are 3 seconds

%%%% the 'star' indicates in spherical coordinates'
% R*: R_star is covariance of the measurement noise

sigmaRgeMtrs = 2*5;         % rge bins / 12 Uniform distr 49/12=4.0833
sigma_r_2 = sigmaRgeMtrs/(1852^2);  % variance of the range estimate (7 m)
%
bear_var = 2*.04;  % .65^2/12 =0.035 Bearing variance based on Uniform distr

sigma_beta_2 = (bear_var * pi/180)^2;
R_star=[          sigma_r_2             0;
                  0                 sigma_beta_2];
```

```
q=10;                  % q gives how much variation we expect in plant.


%%%%%%%%%%%%%%%%%%% LOAD AND PREP DATA  %%%%%%%%%%
%  Load the data set and strip off the time and initialization pts

load ShipDTrkData%  data set of lat, lon & time from simulink

xypos_ship1 = [lat , lon];
timeShip1=time;
rows1 = size(xypos_ship1,1)
shipSize1 = 1200%  8:1 is rule of thumb 80m x 10m

load ShipCTrkData

xypos_ship2 = [lat , lon];
timeShip2=time;
rows2 = size(xypos_ship2,1)
shipSize2 = 3200%  8:1 is rule of thumb 160m x 20m


% pad the data to create equal length vectors
if size(xypos_ship1,1) < size(xypos_ship2,1)
          disp('Ship 2 has longest Track')
          longestTrk = xypos_ship2;
          xypos_ship1 = [xypos_ship1 ;...
                    [0*ones(abs((size(xypos_ship2,1) - ...
          size(xypos_ship1,1))),1)*xypos_ship1(size(xypos_ship1,1),1),...
                    0*ones(abs((size(xypos_ship2,1) -...
          size(xypos_ship1,1))),1)*xypos_ship1(size(xypos_ship1,1),2)]];
else
          disp('Ship 1 has longest Track')
          longestTrk = xypos_ship1;

          xypos_ship2 = [xypos_ship2 ;...
                    [0*ones(abs((size(xypos_ship2,1) - ...
                    size(xypos_ship1,1))),1)*xypos_ship2(size(xypos_ship2,1),1),...
                    0*ones(abs((size(xypos_ship2,1) - ...
                    size(xypos_ship1,1))),1)*xypos_ship2(size(xypos_ship2,1),2)]];
end
          disp('Ship 1 has longest Track')
          longestTrk = xypos_ship1;

          xypos_ship2 = [xypos_ship2 ;...
                    [0*ones(abs((size(xypos_ship2,1) - ...
                    size


%%%%%%%%%  Set Radar Locations  %%%%%%%%%%%%%

rdr1x = -01- 05.53583/60;;%displacement of radar 1 in lat & lon
rdr1y = 41+18.59502/60;      % Governors Island

rdr2x = -05 - 25.33414/60;%displacement of radar 2 Bank Street
rdr2y = 38+48.5168/60 ;

%%%%%%%%Repeat nloop times  %%%%%%%%%%%%%%%

nloop = 1
for ctr = 1:nloop
ctr

          %%%%%%%%      Pseudo-Measurement Vector in X & Y   %%%%%%%%%%%%%%%
          %%%%%%%%    for ships 1 (1) & 2 (2) and rdrs 1 & 2 %%%%%%%%%%%%%%%%
          %
          %          This is where the error is added to simulate the parameters
          %          of the two radars
% What is returned is:
%       z_w_noiseTrk1Rdr1- the 'noisy' lat and lon for track 1 as
%                                                       seen by radar
%       zstar11                         - the 'noisy' range and bearing measurement
%       xypos1                          - the actual lat and lon from simulink

%       Track 1 as seen by RADAR 1: Governors's Island +++++++++++++++++
[z_w_noiseTrk1Rdr1,zstar11,xypos1] =...
                        PseudoMeasure(xypos_ship1,rdr1x,rdr1y,R_star);


%       Track 2 as seen by RADAR 1: Governors's Island +++++++++++++++++
[z_w_noiseTrk2Rdr1,zstar21,xypos2] =...
                        PseudoMeasure(xypos_ship2,rdr1x,rdr1y,R_star);

%       Track 2 as seen by RADAR 2: Bank Street +++++++++++++++++++++++++
[z_w_noiseTrk1Rdr2,zstar12,xypos1] =...
        PseudoMeasure(xypos_ship1,rdr2x,rdr2y,R_star);

%       Track 2 as seen by RADAR 2: Bank Street +++++++++++++++++++++++++
[z_w_noiseTrk2Rdr2,zstar22,xypos2] =...
        PseudoMeasure(xypos_ship2,rdr2x,rdr2y,R_star);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       State Initialization:
%       Using the first two radar returns in x and y form.
%       Compute the initial estimate xhat and phat
%       Let rdr 1 pickup ship 1 first and rdr2 ship 2


[xhat1,phat1,H,F,Qk]=...
        Init_PosnVel(zstar11(1:2,:),z_w_noiseTrk1Rdr1(1:2,:),delta_t,q,R_star);

[xhat2,phat2,H,F,Qk]=...
        Init_PosnVel(zstar22(1:2,:),z_w_noiseTrk2Rdr2(1:2,:),delta_t,q,R_star);


%% Initialize xyhat
% xyhat is xhatk+1 before actual measurement (prediction)
% transposed to match form of xypos--positions from simulink

xyhat1 = (H*inv(F) * xhat1)';

xyhat1 = [ xyhat1               ;
        (H*xhat1)'];

xyhat2 = (H*inv(F) * xhat2)';

xyhat2 = [ xyhat2               ;
                (H*xhat2)'];



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Cycle through the data

n=size(longestTrk,1);%n=size(xypos1,1);

for count=3:n           % start after 2nd obsn to last obsn

        %%%%%%% Nearest Neighbour calculation to determine which measurement
        %                       should be assigned to which track

        %       Nearest Neighbour radar 1
```

```
            ztilde21 = z_w_noiseTrk2Rdr1(count,:)'-H*xhat1;
                    dist21=ztilde21'*ztilde21;

            ztilde22 = z_w_noiseTrk2Rdr1(count,:)'-H*xhat2;
                    dist22=ztilde22'*ztilde22;

            if dist22 <= dist21;

                    zwn11 = z_w_noiseTrk1Rdr1(count,:);
                    zwn21 = z_w_noiseTrk2Rdr1(count,:);

            else

                    zwn11 = z_w_noiseTrk2Rdr1(count,:);
                    zwn21 = z_w_noiseTrk1Rdr1(count,:);

            end


%           Nearest Neighbour radar 2

            ztilde11 = z_w_noiseTrk1Rdr2(count,:)'-H*xhat1;
                    dist11=ztilde11'*ztilde11;

            ztilde12 = z_w_noiseTrk1Rdr2(count,:)'-H*xhat2;
                    dist12=ztilde12'*ztilde12;

            if dist11 <= dist12;

                    zwn12 = z_w_noiseTrk1Rdr2(count,:);
                    zwn22 = z_w_noiseTrk2Rdr2(count,:);

            else

                    zwn12 = z_w_noiseTrk2Rdr2(count,:);
                    zwn22 = z_w_noiseTrk1Rdr2(count,:);

            end




%%%%%%% Prediction Step:  %%%%%% Kalman prediction algorithm

            [xhat1,phat1] = KalmanPredict(xhat1,phat1,F,Qk);
            [xhat2,phat2] = KalmanPredict(xhat2,phat2,F,Qk);


%%%%%%% Update Step:  %%%%%% Kalman Measurement Update Step:
%
%           Computes new position using Kalman gain
%           and the innovations
%           This doing the filtering entirely in x&y coords

r1=         zstar11(count,1);
beta1=      zstar11(count,2);
Fr1=[       cos(beta1), -r1*sin(beta1);
            sin(beta1), r1*cos(beta1)];

r2=         zstar22(count,1);
beta2=      zstar22(count,2);
Fr2=[       cos(beta2), -r2*sin(beta2);
            sin(beta2), r2*cos(beta2)];

% Now convert covariance back to x&y
%           R =         [ sigma_xx^2sigma_xy^2]
%               [ sigma_xy^2sigma_yy^2]

R1 =    Fr1*R_star*Fr1';
R2 =    Fr2*R_star*Fr2';

if Radar == 1
%       Perform measurement update using radar 1 info first
%       This will do the updates based on what Governor's Island sees

        [xhat1,phat1] = KalmanUpdate(xhat1,phat1,H,R1,zwn11');

        [xhat2,phat2] = KalmanUpdate(xhat2,phat2,H,R2,zwn21');

elseif Radar == 2

%       Now perform measurement update again using radar 2 info
%       This will do the updates based on what Bank Street sees

        [xhat1,phat1] = KalmanUpdate(xhat1,phat1,H,R1,zwn12');
        [xhat2,phat2] = KalmanUpdate(xhat2,phat2,H,R2,zwn22');

end     % end which radar to use



% Append to Estimate Matrix

if count <= rows1
   xyhat1 = [xyhat1;(H*xhat1)'];

  if count >= 20% take the crse and speed over 1 min =3 sec x20
   crse1 =  [crse1;math2nautDeg(...
                         (atan2((xyhat1((count-2),2)-xyhat1((count-19),2)),...
                                 (xyhat1((count-2),1)-xyhat1((count-19),1))))+...
                         (atan2((xyhat1((count-1),2)-xyhat1((count-18),2)),...
                                 (xyhat1((count-1),1)-xyhat1((count-18),1))))+...
                         (atan2((xyhat1(count,2)-xyhat1((count-17),2)),...
                                 (xyhat1(count,1)-xyhat1((count-17),1))))*180/(3*pi))];

  speed1 = [speed1; sqrt(  ( ( (xyhat1((count-2),1)-xyhat1((count-19),1)) +...
                   (xyhat1(count-1,1)-xyhat1((count-18),1))+...
                   (xyhat1(count,1)-xyhat1((count-17),1)))/3 ...
                                                                        ).^2 + ...
             ( (       (xyhat1((count-2),2)-xyhat1((count-19),2)) +...
                   (xyhat1(count-1,2)-xyhat1((count-18),2))+...
                   (xyhat1(count,2)-xyhat1((count-17),2)))/3 ...
                                 ).^2)*3600 /(17*delta_t)];% speed in knots
  else
   crse1  = [crse1;NaN];
   speed1 = [speed1;NaN];

  end

end

if count <= rows2
        xyhat2 = [xyhat2;(H*xhat2)'];

 if count >= 20% take the crse and speed over 1 min =3 sec x20
  crse2 =  [crse2;math2nautDeg(...
                         (atan2((xyhat2((count-2),2)-xyhat2((count-19),2)),...
                                 (xyhat2((count-2),1)-xyhat2((count-19),1))))+...
                         (atan2((xyhat2(count-1,2)-xyhat2((count-18),2)),...
                                 (xyhat2(count-1,1)-xyhat2((count-18),1))))+...
                         (atan2((xyhat2(count,2)-xyhat2((count-17),2)),...
                                 (xyhat2(count,1)-xyhat2((count-17),1))))*180/(3*pi))];
 speed2 = [speed2;  sqrt(  ( ( (xyhat2((count-2),1)-xyhat2((count-19),1)) +...
                              (xyhat2(count-1,1)-xyhat2((count-18),1))+...
                              (xyhat2(count,1)-xyhat2((count-17),1)))/3 ...
                                                                                    ).^2 + ...
                     ( (      (xyhat2((count-2),2)-xyhat2((count-19),2)) +...
                              (xyhat2(count-1,2)-xyhat2((count-18),2))+...
                              (xyhat2(count,2)-xyhat2((count-17),2)))/3 ...
                                        ).^2)*3600 /(17*delta_t)];% speed in knots

            else
             crse2  = [crse2;NaN];
             speed2 = [speed2;NaN];

            end


           end
      end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Construct Track Tables for Radars 1 and 2

if Radar == 1

      TrackTableTrk1Rdr1 = [xyhat1 , crse1,speed1,shipSize1*ones(length(timeShip1),1),...
            1003*ones(length(timeShip1),1),timeShip1,...
            0*ones(length(timeShip1),1),9*ones(length(timeShip1),1)];

      TrackTableTrk2Rdr1 = [xyhat2 , crse2,speed2,shipSize2*ones(length(timeShip2),1),...
            1004*ones(length(timeShip2),1),timeShip2,...
            0*ones(length(timeShip2),1),9*ones(length(timeShip2),1)];

      TrackTableRdr1b = [TrackTableTrk1Rdr1;TrackTableTrk2Rdr1];

      % strip off values outside of Radar 1 Range
      % Strip off South of 39'N
      TrackTableRdr1b = TrackTableRdr1b(find(TrackTableRdr1b(:,2)>=39),:);

      % Strip off West of 04'W
      TrackTableRdr1b = TrackTableRdr1b(find(TrackTableRdr1b(:,1)>=-04),:);

%#################################################################
      save TrackTableGIb TrackTableRdr1b
%#################################################################

elseif Radar == 2

      TrackTableTrk1Rdr2 = [xyhat1 , crse1,speed1,shipSize1*ones(length(timeShip1),1),...
            2003*ones(length(timeShip1),1),timeShip1,...
            0*ones(length(timeShip1),1),9*ones(length(timeShip1),1)];

      TrackTableTrk2Rdr2 = [xyhat2 , crse2,speed2,shipSize2*ones(length(timeShip2),1),...
            2004*ones(length(timeShip2),1),timeShip2,...
            0*ones(length(timeShip2),1),9*ones(length(timeShip2),1)];

      TrackTableRdr2b = [TrackTableTrk1Rdr2;TrackTableTrk2Rdr2];

      % strip off values outside of Radar 1 Range
      % Strip off North of 40.5'N
      TrackTableRdr2b = TrackTableRdr2b(find(TrackTableRdr2b(:,2)<=40.5),:);

      % Strip off East of 02'W
      TrackTableRdr2b = TrackTableRdr2b(find(TrackTableRdr2b(:,1)<=-02),:);

%#################################################################
      save TrackTableBankb TrackTableRdr2b
```

```
%##################################################################

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Compute Errors

% error is actual position minus estimated position

poserr1= xypos_ship1(1:rows1,:)-xyhat1;
poserr2= xypos_ship2(1:rows2,:)-xyhat2;

% covariance matrix of the innovations

sse1= diag(poserr1*poserr1');
sse2= diag(poserr2*poserr2');

%%%     Keep track of mean distance errors in disterr1
disterr1 = sqrt(sse1);
if ctr ==1
        distmean1 = disterr1/nloop;
        xyhatmean1 = xyhat1/nloop;

else
        distmean1 = distmean1 + disterr1/nloop;
        xyhatmean1 = xyhatmean1 + xyhat1/nloop;
end

%%%     Keep track of mean distance errors in disterr1
disterr2 = sqrt(sse2);
if ctr ==1
        distmean2 = disterr2/nloop;
        xyhatmean2 = xyhat2/nloop;

else
        distmean2 = distmean2 + disterr2/nloop;
        xyhatmean2 = xyhatmean2 + xyhat2/nloop;
end

end     % end of outside loop


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Plots

if PlotMe > 1



figure(1); clg
showChart3

if 0
 plot(z_w_noiseTrk1Rdr1(:,1),z_w_noiseTrk1Rdr1(:,2),'m')
 plot(z_w_noiseTrk1Rdr2(:,1),z_w_noiseTrk1Rdr2(:,2),'r')
 plot(z_w_noiseTrk2Rdr1(:,1),z_w_noiseTrk2Rdr1(:,2),'b')
 plot(z_w_noiseTrk2Rdr2(:,1),z_w_noiseTrk2Rdr2(:,2),'c')
end
plot(xyhat1(:,1),xyhat1(:,2),'b*')
 plot(xyhat2(:,1),xyhat2(:,2),'r+')

plot(xyhat1(:,1),xyhat1(:,2),'y.')
 plot(xyhat2(:,1),xyhat2(:,2),'g.')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2); clg
hold on
%plot(...
%timeShip1,disterr1(1:size(timeShip1,1))*1852,'b-',...
%timeShip2,disterr2(1:size(timeShip2,1))*1852,'r-');

plot(...
timeShip1(1:634,:),disterr1(1:634)*1852,'b:',...
timeShip2(1:634,:),disterr2(1:634)*1852,'r-');

fig2=gcf;
child = get(fig2,'Children');
 placeMe=get(child,'Position');


Est_err_title=['Radar ',num2str(Radar), ' On: Est Dist Error for Ships 1 & 2: q = ',...
 num2str(q), ' & bearing var = ', num2str(bear_var), ' degrees'];

title(Est_err_title);
%axis([0,rows,0,40])
xlabel('time step = 3 sec')
ylabel('Miss Distance (m)')
grid


h=legend( 'Single Run Distance Error Ship 1',...
'Single Run Distance Error Ship 2');

PutMe=get(h,'Position');
set(h,'Position',[placeMe(1),(placeMe(2)+placeMe(4)-PutMe(4)),PutMe(3),PutMe(4)])
axes(h)

hold off



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%     Plot the actual and estimated track

figure(3);
clg

showChart1
hold on


plot(       xyhat1(:,1),xyhat1(:,2),'b-',...
            xyhat2(:,1),xyhat2(:,2),'r-')

plot(xypos1(:,1),xypos1(:,2),'c-.',...
            xypos2(:,1),xypos2(:,2),'m-.')


hold off

%Est_posn_title1=['Radar ',num2str(Radar), ' On: Actual and Est Trajectories ',...
%'; q = ', num2str(q), ' & bearing var = ', num2str(bear_var), ...
%' degrees'];



%title(Est_posn_title1)
%xlabel('x distance (nm or min)')
%ylabel('y distance (nm or min)')
```

```
fig3=gcf;
child = get(fig3,'Children');
 placeMe=get(child(1),'Position');

h7=legend('Actual trajectory Ship 1', 'Est Trajectory Ship 1',...
          'Actual trajectory Ship 2', 'Est Trajectory Ship 2');
set(h7,'FontSize',6)

PutMe=get(h7,'Position');
set(h7,'Position',[(placeMe(1)+placeMe(3)-PutMe(3)),...
(placeMe(2)+.02),PutMe(3),PutMe(4)])
axes(h7)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(4);clg
title('Course and Speed From Kalman Filter')

subplot(2,2,1); plot(timeShip1(1:634,:),crse1(1:634,:),'b');
xlabel('Time (sec)'); ylabel('Course (degrees)')

subplot(2,2,2); plot(timeShip2(1:634,:),crse2(1:634,:),'r')
xlabel('Time (sec)'); ylabel('Course (degrees)')

subplot(2,2,3); plot(timeShip1(1:634,:),speed1(1:634,:),'b')
axis([21,timeShip1(length(timeShip1)),8.5,11.5])
xlabel('Time (sec)'); ylabel('Speed (knots)')

subplot(2,2,4); plot(timeShip2(1:634,:),speed2(1:634,:),'r')
axis([21,timeShip2(length(timeShip2)),8.5,11.5])
xlabel('Time (sec)'); ylabel('Speed (knots)')


end % PlotMe >1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%     Plot the estimated data track to feed into the fusion algorithm

if PlotMe >= 1

figure(5);
clg

showChart1

if exist('TrackTableRdr1b')
          plot(TrackTableRdr1b(:,1),TrackTableRdr1b(:,2),'bo')
end


if  exist('TrackTableRdr2b')
          plot(TrackTableRdr2b(:,1),TrackTableRdr2b(:,2),'r+')
end

if exist('TrackTableRdr1b')
          plot(TrackTableRdr1b(:,1),TrackTableRdr1b(:,2),'y.')
end
if  exist('TrackTableRdr2b')
          plot(TrackTableRdr2b(:,1),TrackTableRdr2b(:,2),'k.')
end

if 1

fig5=gcf;
child = get(fig5,'Children');
 placeMe1=get(child(1),'Position');

h8=legend('Radar 1 Tracks', ...
          'Radar 2 Tracks');
```

```
set(h8,'FontSize',6)

PutMe1=get(h8,'Position');
set(h8,'Position',[(placeMe1(1)+placeMe1(3)-PutMe1(3)),...
(placeMe1(2)+.02),PutMe1(3),PutMe1(4)])
axes(h8)
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(6);
clg

showChart3

if exist('TrackTableRdr1b')
        plot(TrackTableRdr1b(:,1),TrackTableRdr1b(:,2),'b.')
end


if  exist('TrackTableRdr2b')
        plot(TrackTableRdr2b(:,1),TrackTableRdr2b(:,2),'r.')
end

if 1

fig6=gcf;
child = get(fig6,'Children');
 placeMe1=get(child(1),'Position');

h9=legend('Radar 1 Tracks', ...
        'Radar 2 Tracks');
set(h9,'FontSize',6)

PutMe1=get(h9,'Position');
set(h9,'Position',[(placeMe1(1)+placeMe1(3)-PutMe1(3)),...
(placeMe1(2)+.02),PutMe1(3),PutMe1(4)])
axes(h9)
end



end %    end PlotMe >=1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




if PrintMe == 1

if Radar == 1

        figure(1)
        print -dpsc fig1EstTrksRdr1b
        print -dgif8 fig1EstTrksRdr1b
        figure(2)
        print  -dpsc fig2ErrorRdr1b
        print  -dgif8 fig2ErrorRdr1b
        figure(3)
        print -dpsc fig3TrajRdr1b
        print -dgif8 fig3TrajRdr1b
        figure(4)
        print -dpsc fig4CrseSpeed1b
        print -dgif8 fig4CrseSpeed1b

elseif Radar ==2
```

```
          figure(1)
          print -dpsc fig1EstTrksRdr2b
          print -dgif8 fig1EstTrksRdr2b
          figure(2)
          print  -dpsc fig2ErrorRdr2b
          print  -dgif8 fig2ErrorRdr2b
          figure(3)
          print -dpsc fig3TrajRdr2b
          print -dgif8 fig3TrajRdr2b
          figure(4)
          print -dpsc fig4CrseSpeed2b
          print -dgif8 fig4CrseSpeed2b

end       % end Radar
          figure(5)
          print -dpsc fig5BothRdrsb
          print -dgif8 fig5BothRdrsb
          figure(6)
          print -dpsc fig6ZoomBothRdrsb
          print -dgif8 fig6ZoomBothRdrsb

end       % end PrintMe
```

PseudoMeasure.m

```
function [z_w_noise,zstar,xypos]=PseudoMeasure(xypos,rdrlocx,rdrlocy,R_star)
%
%        PseudoMeasure.m
%        Major Ian Glenn
%
%        Written by:         Major Ian Glenn
%
%        Created:            14 Oct 95
%        Modified:           17 Oct 95
%
%        MULTI-SENSOR, MULTI-TARGET TRACKING
%        OF SHIPPING TRAFFIC IN NEW YORK HARBOUR
%        BY THE VESSEL TRAFFIC SYSTEM.
%
%        Take xypos data from simulink with the radar location
%        and create pseudomeasurements in x&y with noise added

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%        Initialize variables
%
%%%% the 'star' indicates in spherical coordinates'
% R*: R_star is covariance of the measurement noise


%%%%%%%%%%%%%%%%%%%% PREP DATA %%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Strip off the time and initialization pts

data = xypos;

%rows = size(xypos,1);

%        Take the x&y position data from  simulink and
%        compute rge and bearing with additive noise (uncertainty)
% z* = [ sqrt( {x-rdrlocx}^2 + {y-rdrlocy}^2)] <= rge + [rge variation ]
%        [ atan2({y-rdrlocy}/{x-rdrlocx})      ] <= bearing [bearing variation]

%%%%%%%%%% Measurement Matrix in RANGE AND BEARING %%%%%%%%%%%%%%%%
 zstar = [ sqrt((xypos(:,1)-rdrlocx).^2 + (xypos(:,2)-rdrlocy).^2) , ...
              atan2((xypos(:,2)-rdrlocy),(xypos(:,1)-rdrlocx)) ] ...
         + [randn(size(xypos,1),1) .* (sqrt(R_star(1,1))) ,...
              randn(size(xypos,1),1) .* (sqrt(R_star(2,2)))];

% check result with "polar(zstar(:,2), zstar(:,1)) "

%figure;
%hold on
%polar(zstar11(:,2), zstar11(:,1))
%hold off


%        Now convert back to a pseudo-measurement x&y
%        Convert rge & bear back to x&y
%        z =      rge * cos(bearing) = [ x ]
%                 rge * sin(bearing)= [ y ]

%%%%%%%%%% Pseudo-Measurement Vector in X & Y %%%%%%%%%%%%%%%%
z_w_noise = [ zstar(:,1) .* cos(zstar(:,2)) , zstar(:,1) .* sin(zstar(:,2))];

z_w_noise = [(z_w_noise(:,1)+rdrlocx),(z_w_noise(:,2)+rdrlocy)];
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% check with"plot(z_w_noise(:,1),z_w_noise(:,2));axis([0 10000 0 10000])"
figure;plot((z_w_noise(:,1)+rdrlocx),(z_w_noise(:,2)+rdrlocy));
axis([0 10000 0 10000])
h3=line([rdrlocx rdrlocx],[rdrlocy-500,rdrlocy+500]);
        set(h3,'Linestyle','--')
        set(h3,'color','green')
        text(rdrlocx+100,rdrlocy+100,'Radar 1')
h4=line([rdrlocx-500 rdrlocx+500],[rdrlocy,rdrlocy]);
        set(h4,'Linestyle','--')
        set(h4,'color','green')
```

KalmanPredict.m

```
function [xhat_new,phat_new] =KalmanPredict(xhat,phat,Phi,Qk);

%       KalmanPredict:      takes the old measurement and covariance
%                           data, and predicts expected value for new

%       Expected value is just Phi matrix times old average value

xhat_new = Phi*xhat;

%       New covariance is increased by the discrete plant noise Qk

phat_new = (Phi*phat*Phi')+Qk;
```

KalmanUpdate.m

```
function [xhat_new,phat_new] = KalmanUpdate(xhat,phat,H,R,z);
%
%       KalmanUpdate:   takes new measurement and old prediction
%                       info and provides updated position estimate
%                       and covariance

%       Calculate Kalman gain

K= phat*H'*inv((H*phat*H'+R));

%       Calculate the innovations

innovations = z-(H*xhat);

%       Update xnew given new measurement

xhat_new = xhat + K*innovations;

%       Update covariance based on new measurement

I=eye(size(K*H));
phat_new = (I-K*H)*phat*(I-K*H)' + K*R*K';
```

Init_PosnVel

```
function [xhat,phat,H,F,Qk]=Init_PosnVel(zstar,z_w_noise,delta_t,q,R_star)

%
%
%       Major Ian Glenn
%
%       POSITION-VELOCITY Initialization Function
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Initialize variables

% F= Phi: state transition matrix from k to k+1
%       This is the consant velocity model
F=[     1       delta_t 0       0;
        0       1       0       0;
        0       0       1       delta_t;
        0       0       0       1];

% H: matrix corresponds to state space C, observed variables
H=[     1 0 0 0;
        0 0 1 0];

% Q: covariance of the plant noise with delta t = .1sec

Sigma1 = [delta_t^3/3 delta_t^2/2;
                delta_t^2/2 delta_t  ];


Qk=q*[  Sigma1          zeros(2,2);
        zeros(2,2)Sigma1  ];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       Estimate Position from data:
%       Using the first two radar returns in rge and bearing form
%       Compute the initial estimate of position from the rge and
%       bearing and express the result as x and y.

CapX = [ H                      ;
                H* inv(F)       ];

z=[     z_w_noise(2,:),z_w_noise(1,:)]';

xposn   =       z(1);
xvel    =       (z(3)-z(1))/delta_t;
yposn   =       z(2);
yvel    =       (z(4)-z(2))/delta_t;

xhat = [ xposn, xvel, yposn, yvel ]';

% now compute phat ( the initial covariance estimate)

        Finv =  inv(F);

        r=      zstar(2,1);
        beta=   zstar(2,2);
        Fr1=[   cos(beta), -r*sin(beta);
                sin(beta), r*cos(beta)];

        R1 =    Fr1*R_star*Fr1';
        r=       zstar(1,1);
        beta=    zstar(1,2);
        Fr2=[    cos(beta), -r*sin(beta);
                 sin(beta), r*cos(beta)];

        R2 =     Fr2*R_star*Fr2';

        E_Z2_Z2t = H*Finv*Qk*Finv'*H' + R2;

E_Z_Zt = [       R1               zeros(2,2);
                 zeros(2,2)       E_Z2_Z2t];

phat =  inv(CapX)*E_Z_Zt*E_Z_Zt'*inv(CapX)';

return
```

APPENDIX C. PREPROCESSING CODE

This appendix contains the code used to preprocess the Telephonics radar data for use in fusion algorithm. The code in this appendix is as follows:

- **ReadData.m** on page 100.
- **hex2bstr.m** on page 103.
- **printPretty.m** on page 104.
- **simpleCoordConv.m** on page 105.
- **naut2mathRad.m** on page 106.
- **rads2DMS.m** on page 107.

ReadData.m

```
%       ReadData.m<PRE>
%
%       By:     Major Ian Glenn
%
%       written:  10 Sep 95
%       modified: 20 Oct 95
%
%       This algorithm is designed to read hex code from radar processor
%       convert it into binary to extract the necessary flags
%       and information.
%
%       Design:
%
%               Initialize variables
%               While not end of file
%                       Build a 2 by n matrix of hex data read from ASCII File
%               end while
%
%               While not end of matrix
%                       Look for header and set pointer
%                       Decode and translate each data segment
%                       Append to data matrix
%               end while

clear

A= ['0';'0'];
ObsnMatrix=zeros(9,1);
nm2metres = 1852.0;


fid=fopen('tracks3.m');% get data from file

%       This assumes that the header will be '8' in the first line
%       and '0' in the second line.

while 1                 % while true read in the file line by line
        line1 = fgetl(fid);
        if ~isstr(line1), break, end

                line1 = line1(1: find(line1=='*')-1);% trim off '*'


        line2 = fgetl(fid);
        if ~isstr(line2), break, end
                line2 = line2(1: find(line2=='*')-1);% trim off '*'

                A       = [A(1,:) ,line1;...
                           A(2,:), line2];

end

fclose(fid);

vectorLength = size(A,2);

row1= findstr(A(1,:),'8');
row2 =  findstr(A(2,:),'0');
ptr = row1;

i=1; j=1;
while i <= length(row1);

        if find(row1(i)==row2) ~= []
```

```
                indx(j)=find(row1(i)==row2)  ;
                i=i+1;
                j=j+1;
        else
                i=i+1;
        end
end


for i=1:   length(row1)

        ptr = row2(indx(i));

        if ptr+36 >= vectorLength, break,end%  stop before exceed matrix dimension



        header              = hex2bstr([A(1,ptr), A(2,ptr)]);
        siteIDno  = hex2bstr([A(1,ptr+1), A(2,ptr+1)]);

%       bstr_dec(siteIDno);
siteIDnoDec =1                  % simulate a site ID number since one is not given
%
        %       once identify the radar site can retrieve it's location and height
        rdrlat=0;rdrlon=0;

        trackIDno= hex2bstr([A(1,ptr+2), A(2,ptr+2), ...
                                        A(1,ptr+3), A(2,ptr+3)]);
        trackIDnoDec = bstr_dec(trackIDno)

        if trackIDnoDec <= 1022% filter out the *filler track*
                                                                % track 1023 is used to keep system
                                                                % filled with data

        trackTime= hex2bstr([A(1,ptr+4), A(2,ptr+4), ...
                                        A(1,ptr+5), A(2,ptr+5), ...
                                        A(1,ptr+6), A(2,ptr+6), ...
                                        A(1,ptr+7), A(2,ptr+7), ...
                                        A(1,ptr+8), A(2,ptr+8)]);

        secondsBin =trackTime(1:32);
        secondsDec=bstr_dec(secondsBin);

        sec100Bin = trackTime(34:40);
        sec100Dec = bstr_dec(sec100Bin);


        crse                = hex2bstr([A(1,ptr+9), A(2,ptr+9), ...
                                        A(1,ptr+10), A(2,ptr+10)]);

        crseDeg= bstr_dec(crse(3:16)) * 360/16384;

        speed               = hex2bstr([A(1,ptr+11), A(2,ptr+11), ...
                                        A(1,ptr+12), A(2,ptr+12)]);

        speedKts= bstr_dec(speed) * 0.0625;

        predRge             = hex2bstr([A(1,ptr+13), A(2,ptr+13), ...
                                        A(1,ptr+14), A(2,ptr+14)]);

        predRgeNM= bstr_dec(predRge) * 256/65536;

        predAz              = hex2bstr([A(1,ptr+15), A(2,ptr+15), ...
                                        A(1,ptr+16), A(2,ptr+16)]);
```

```
            predAzDeg= bstr_dec(predAz(3:16)) * 360/16384;

            rdrRge          = hex2bstr([A(1,ptr+17), A(2,ptr+17), ...
                                        A(1,ptr+18), A(2,ptr+18)]);

            rdrRgeNM= bstr_dec(rdrRge) * 256/65536;

            rdrAz           = hex2bstr([A(1,ptr+19), A(2,ptr+19), ...
                                        A(1,ptr+20), A(2,ptr+20)]);

            rdrAzDeg= bstr_dec(rdrAz(3:16)) * 360/16384;

            %       calculate the latitude and longitude of the target

            [tgtlat,tgtlon]=simpleCoordConv(rdrlat,rdrlon,predRgeNM,radians(predAzDeg));

            extentRge= hex2bstr([A(1,ptr+21), A(2,ptr+21)]);

            extentRgeNM= bstr_dec(extentRge) * 256/65536;

            extentAz  = hex2bstr([A(1,ptr+22), A(2,ptr+22), ...
                                        A(1,ptr+23), A(2,ptr+23)]);

            extentAzDeg= bstr_dec(extentAz(3:16)) * 360/16384;

            vesselSize = (extentRgeNM * nm2metres)*...
                                        2* predRgeNM*nm2metres*tan( extentAzDeg*pi/360 );

            trackQual= hex2bstr([A(1,ptr+24), A(2,ptr+24)]);

            if trackQual(1) == '1'; autoTrk=1; else; autoTrk=0;end
            if trackQual(2) == '1'; manualTrk=1; else; manualTrk = 0;end
            if trackQual(3) == '1'; lostTgt=1; else; lostTgt = 0;end
            if trackQual(4) == '1'; coastTgt=1; else; coastTgt = 0;end
            trackQualDec= bstr_dec(trackQual(5:8));

            % The following Data Structure will be used throughout this algorithm:
            %                               1       Tgt Latitude - X Position  [DDMM.SSSSSS]
            %                               2       Tgt Longitude - Y Position[DDMM.SSSSSS]
            %                               3       Course - COG - Crse over Ground[degrees]
            %                               4       Speed - SOG - Speed over Ground[knots]
            %                               5       Size - This is Extent Rge (m) x Extent Az (m)[m^2]
            %                               6       Site Id - This is the reporting radar in the 1000th posn
            %                                                       and the given track number
            %                               7       Time - This the GPS time in seconds from system start[sec]
            %                               8       Lost Track - set when the Kalman filter fails
            %                               9       Track Quality - 0-9
            %                           10      PlatformNo - Global Track Name associated with a platform icon

            ObsnMatrix=[ObsnMatrix(1,:) , tgtlat;...
                                ObsnMatrix(2,:) , tgtlon ;...
                                ObsnMatrix(3,:) , crseDeg ;...
                                ObsnMatrix(4,:) , speedKts ;...
                                ObsnMatrix(5,:) , vesselSize;...
                                ObsnMatrix(6,:) , (siteIDnoDec *1000+trackIDnoDec);...
                                ObsnMatrix(7,:) , (secondsDec +  sec100Dec/100) ;...
                                ObsnMatrix(8,:) , lostTgt ;...
                                ObsnMatrix(9,:) , trackQualDec];

            % note:  At this point the site is multipled by 1000 and combined with
            %                       the track number to make a composite identifier
            %                       i.e. site 1 and track 2 becomes 1002
            %                       This is possible as the max number of tracks per site is 999.
            end             % end track filler condition
    end
    ObsnMatrix=ObsnMatrix(:,2:size(ObsnMatrix,2));% trim off initial zero column
    ObsnMatrix=ObsnMatrix';   % reorient the vector
    printPretty(ObsnMatrix);
```

hex2bstr.m

```
function S=hex2bstr(x);
%
%        Maj Ian Glenn
%        inglenn@nps.navy.mil
%        last rev 13 Jul 95
%
%        converts a hex number into a binary
%        number in a string vector S.
%
S=[];
n=x;
%        n=dec2hex(x);
for i=1:length(n);

        if strcmp(n(1,i),'F')
                ssub='1111';
        elseif strcmp(n(1,i),'E')
                ssub='1110';
        elseif strcmp(n(1,i),'D')
                ssub='1101';
        elseif strcmp(n(1,i),'C')
                ssub='1100';
        elseif strcmp(n(1,i),'B')
                ssub='1011';
        elseif strcmp(n(1,i),'A')
                ssub='1010';
        elseif strcmp(n(1,i),'9')
                ssub='1001';
        elseif strcmp(n(1,i),'8')
                ssub='1000';
        elseif strcmp(n(1,i),'7')
                ssub='0111';
        elseif strcmp(n(1,i),'6')
                ssub='0110';
        elseif strcmp(n(1,i),'5')
                ssub='0101';
        elseif strcmp(n(1,i),'4')
                ssub='0100';
        elseif strcmp(n(1,i),'3')
                ssub='0011';
        elseif strcmp(n(1,i),'2')
                ssub='0010';
        elseif strcmp(n(1,i),'1')
                ssub='0001';
        else ssub='0000';

        end

S=[S ssub];

end
```

printPretty.m

```
function []=printPretty(InputMatrix)
%       function []=printPretty(InputMatrix)
%
%       by Major Ian Glenn
%
%       written:        9 Sep
%       mod:            19 Oct 95
%
%       Design:
%
%       Takes the track matrix and formats it in a more readable form.



disp('LatitudeLongitude    Course    Speed    Size    '...
'Site &     Time    LT  TQ')
disp(' (nm)    (nm)       (deg)     (kts)    (m^2)    '...
'Track      (sec)     ')

%                         lat          lon         crse   spd   siz        site   time  lt   tq
out= sprintf('%10.7f %12.7f %7.1f  %7.2f %8.1f %12.0f %12.2f %4.0f %4.0f  \n',InputMatrix');

disp(out)
```

simpleCoordConv.m

```
function [tgtlat,tgtlon]=simpleCoordConv(rdrlat,rdrlon,Rnm,mathRad)

% function [tgtlat,tgtlon]=simpleCoordConv(rdrlat,rdrlon,Rnm,mathRad)
%
%       Written by:  Major Ian Glenn
%
%       Created:            5 Sep 95
%       Modified:           20 Oct 95
%
%       Input:
%               I1                      - rdrlat    Radar site latitude in DDMM.SS
%   I2                  - rdrlon    Radar site longitude in DDMM.SS
%   I3                  - Rnm               Rge in NM from radar site
%   I4                  - mathRadMathematical radian bearing from radar site
%
%       Output:
%               O1                      - tgtlat    Target latitude in MM.SS
%   O2                  - tgtlon    Target longitude in MM.SS
%
%
%       Design:
%
%               This function provides simplified conversion of lat and lon from RSP
%
%       Calls:
%               naut2mathRad.m   -      converts nautical reference to mathematical ref
%               rads2DMS.m              -       convert radians to degrees,minutes and seconds
%
%
%

NMperDEG = 60;
NMperRAD = NMperDEG*180/pi;

%disp( num2str(mathRad*180/pi))

A = naut2mathRad(mathRad); % convert from nautical frame of reference to math ref

tgtlatRad = rdrlat + (Rnm * cos(A) / NMperRAD);

tgtlonRad = rdrlon + (Rnm * sin(A) / (NMperRAD * cos(rdrlat)));

%  disp( 'RSP SUCCESS' );


%disp( 'Tgt Latitude')
%disp( num2str(tgtlatRad))
[tgtdeg, tgtmin, tgtsec]=rads2DMS(tgtlatRad);
tgtlat= tgtdeg*100+tgtmin+tgtsec/60

%disp( 'Tgt Longitude')
%disp( num2str(tgtlonRad))
[tgtdeg, tgtmin, tgtsec]=rads2DMS(tgtlatRad);
tgtlon = tgtdeg*100+tgtmin+tgtsec/60
```

naut2mathRad.m

```
function [mathRad]=naut2mathRad(nautRad)
% convert nautical based coord system to math system
% in radians fro computations
%
%	function [mathRad]=naut2mathRad(nautRad)
%
%	mod: 21 Aug 95

if nautRad > 2*pi
	error('input greater than 2 Pi radians')
end

if nautRad >= pi/2
	mathRad = 2*pi -nautRad+pi/2;
	else
	mathRad = pi/2 - nautRad;
end
% rotate by changing direction
```

rads2DMS.m

```
function [tgtdeg, tgtmin, tgtsec]=rads2DMS(tgtrads)

% function          [tgtdeg, tgtmin, tgtsec]=rads2DMS(tgtrads)
%
%           Written by:  Major Ian Glenn
%
%           Created:                21 Aug 95
%           Modified:               20 Oct 95
%
%           Input:
%                       I1                              - tgtrads
%
%           Output:
%                       O1                              - tgtdeg
%                       O2                              - tgtmin
%                       O3                              - tgtsec
%
%           Design:
%
%                       This function takes rads and returns DMS for plotting on chart
%
%
%           Calls:
%            .m                 -


tgtdegf=tgtrads*180/pi;

tgtdeg = fix(tgtdegf);

tgtminf = tgtdegf - tgtdeg;

tgtminf = (tgtdegf - tgtdeg)*60;

tgtmin = fix(tgtminf) ;

tgtsec = (tgtminf - tgtmin)*60;

if tgtsec > 0.0001    % Trim precision
            tgtsec = tgtsec;
            else
            tgtsec =0;
end


%Out= [ num2str(tgtdeg, 0),' / ', num2str(tgtmin,0),' / ',...
%                       num2str(tgtsec,5),'  (Degrees/Min/Sec)  >>> note: 1 min = 1 nm'];
%disp(Out)
```

APPENDIX D. SIMULINK SIMULATION PARAMETERS

This appendix contains the descriptions of the Simulink model used to generate the trajectories used in the Multitarget Kalman Filter. Figure D.1 shows the model. The Matlab program, **turn_ms1.m** on page 111, was used in the Matlab Function Block. The other block parameters are detailed in Figure D.2 and Figure D.3 .

Table D.1 shows the parameters used in **ShipCInitSim.m** on page 114 to generate the trajectories.

Parameters	**Ship A**	**Ship B**	**Ship C**	**Ship D**
Initial Latitude (min.sec)	-2.00	-5.46	-5.46	-0.45
Initial Longitude (min.sec)	41.90	38.89	38.89	41.80
vesselVel (m/s)	0.0028	0.0028	0.0028	0.0028
crse (nautical degrees)	180	68	68	200
startTurn1 (sec)	260	122	122	142
endTurn1 (sec)	440	330	330	300
initialValueTurn1 (rad/sec)	0	0	0	0
finalValueTurn1 (rad/sec)	-w	-w	-w	-w
startTurn2 (sec)	1160	480	400	1200
endTurn2 (sec)	1340	680	750	1450
initialValueTurn2 (rad/sec)	0	0	0	0
finalValueTurn2 (rad/sec)	w	-w	w	w
endtime (sec)	1700	900	1900	2050
Note: turn rate, w, was set to 0.0044 rad/sec for all simulations				

Table D.1: Parameters Used To Generate Trajectories

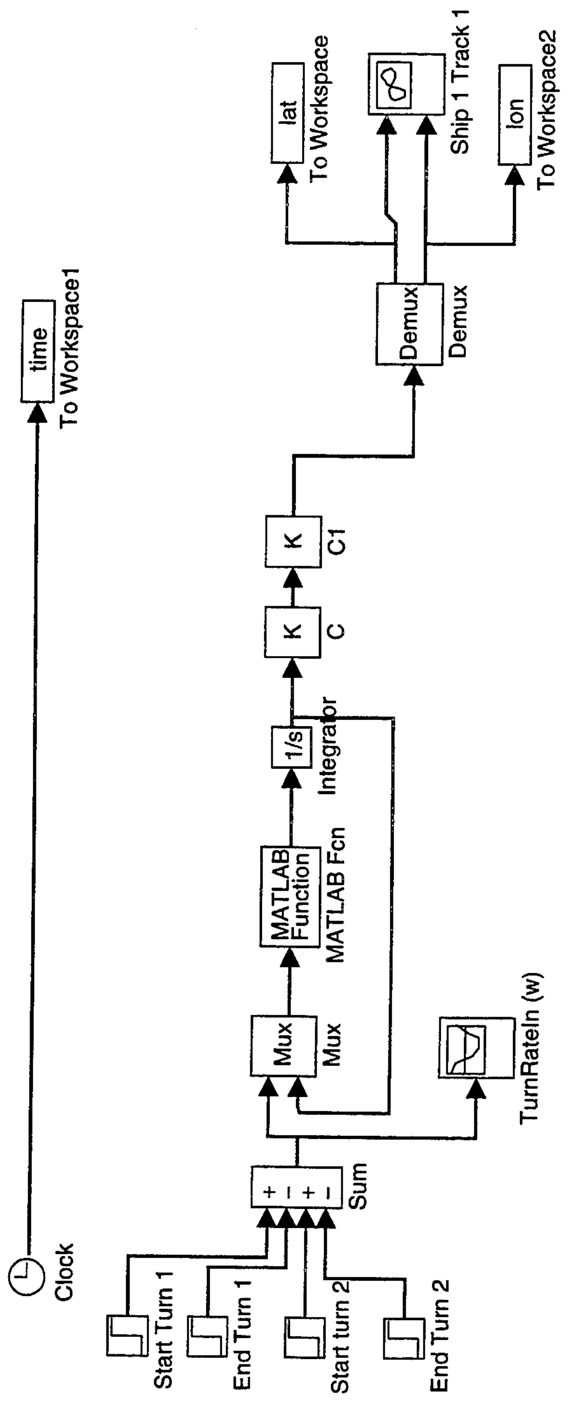
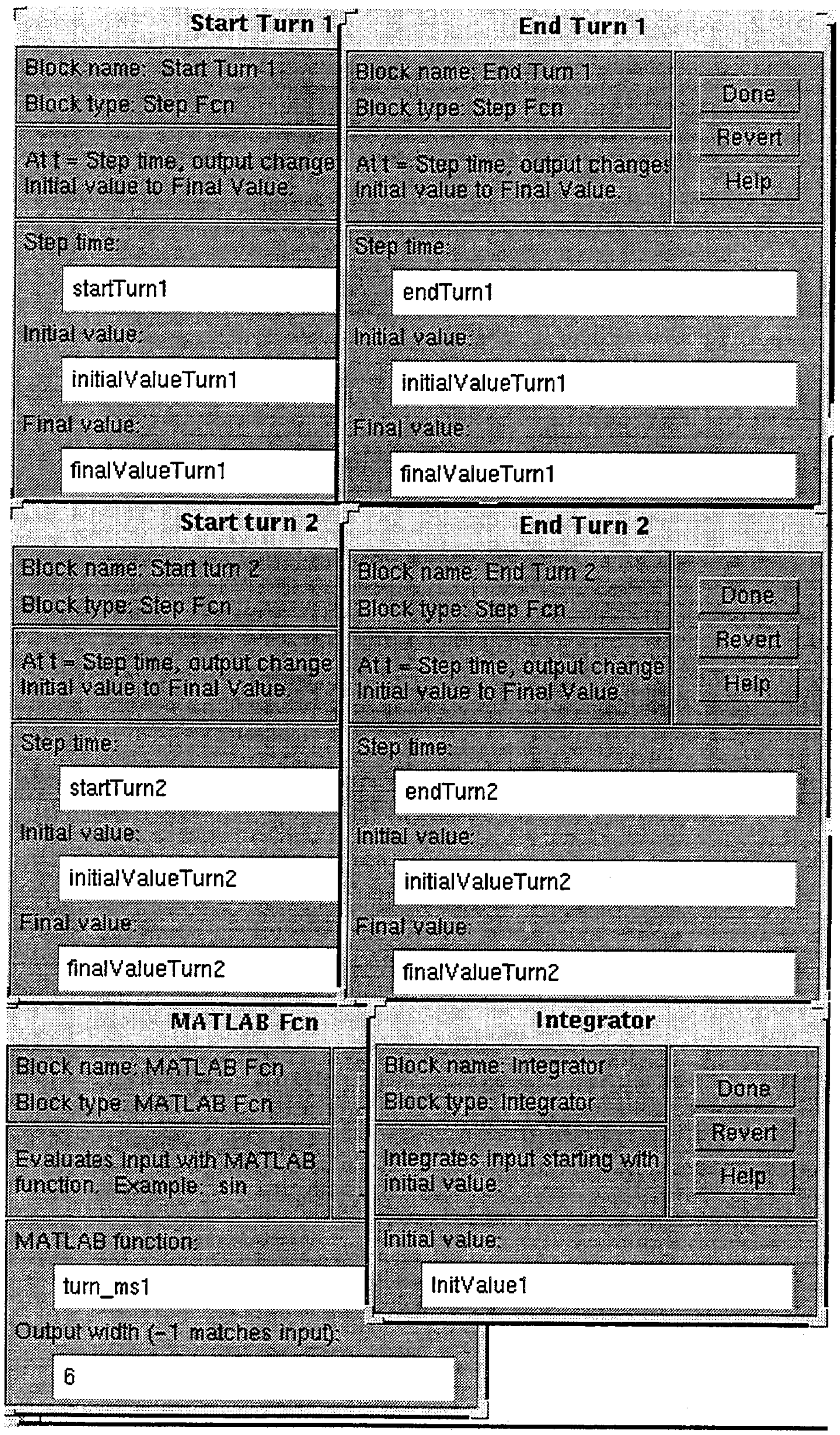


Figure D.1. Simulink Model Used To Generate Trajectories

turn_ms1.m

```
function x0 = turn(u)
%        for maneuver simulation 1

%        Define plane of turn with normal vector, P= [x;y;z]

p= [0;0;1];% defines turn in xy plane

w = u(1); % turn rate

x = u(2:7);% state vector is contained here

F =      [        0 1             0 0             0 0;
                  0 0             0 -w*p(3)       0 w*p(2);
                  0 0             0 1             0 0;
                  0 w*p(3)        0 0             0 -w*p(1);
                  0 0             0 0             0  1;
                  0 -w*p(2)       0 w*p(1)        0 0];

x0 = F*x;
```



Start Turn 1

Block name: Start Turn 1
Block type: Step Fcn

At t = Step time, output change Initial value to Final Value.

Step time: startTurn1
Initial value: initialValueTurn1
Final value: finalValueTurn1

End Turn 1

Block name: End Turn 1
Block type: Step Fcn

At t = Step time, output changes Initial value to Final Value.

Step time: endTurn1
Initial value: initialValueTurn1
Final value: finalValueTurn1

Start turn 2

Block name: Start turn 2
Block type: Step Fcn

At t = Step time, output change Initial value to Final Value.

Step time: startTurn2
Initial value: initialValueTurn2
Final value: finalValueTurn2

End Turn 2

Block name: End Turn 2
Block type: Step Fcn

At t = Step time, output change Initial value to Final Value.

Step time: endTurn2
Initial value: initialValueTurn2
Final value: finalValueTurn2

MATLAB Fcn

Block name: MATLAB Fcn
Block type: MATLAB Fcn

Evaluates input with MATLAB function. Example: sin

MATLAB function: turn_ms1
Output width (-1 matches input): 6

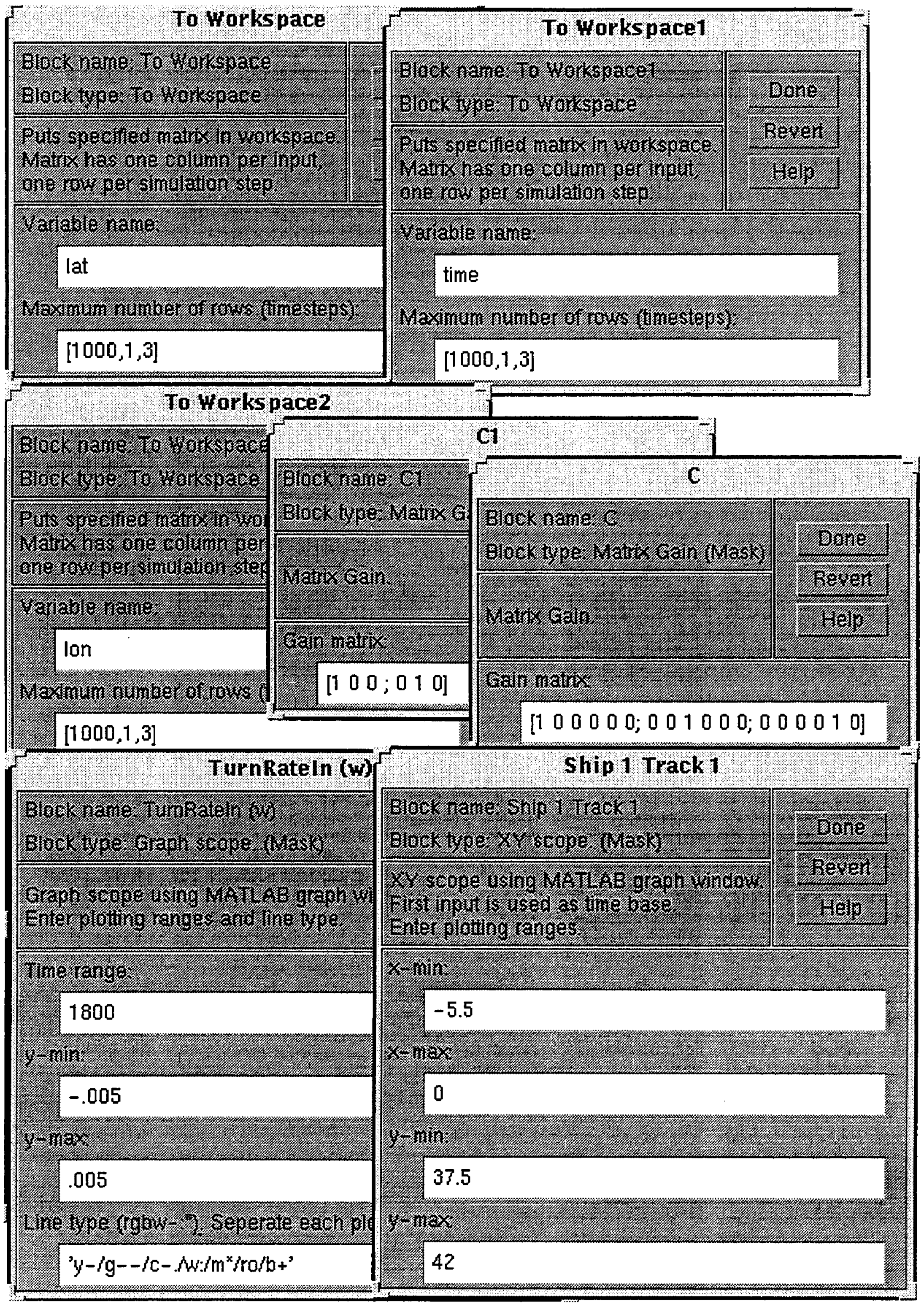
Integrator

Block name: Integrator
Block type: Integrator

Integrates input starting with initial value.

Initial value: InitValue1

Figure D.2. Simulink Model Parameters One



To Workspace

Block name: To Workspace
Block type: To Workspace

Puts specified matrix in workspace.
Matrix has one column per input,
one row per simulation step.

Variable name:
lat

Maximum number of rows (timesteps):
[1000,1,3]

To Workspace1

Block name: To Workspace1
Block type: To Workspace

Puts specified matrix in workspace.
Matrix has one column per input,
one row per simulation step.

Variable name:
time

Maximum number of rows (timesteps):
[1000,1,3]

To Workspace2

Block name: To Workspace
Block type: To Workspace

Puts specified matrix in wo
Matrix has one column per
one row per simulation step

Variable name:
lon

Maximum number of rows (
[1000,1,3]

C1

Block name: C1
Block type: Matrix G

Matrix Gain.

Gain matrix:
[1 0 0 ; 0 1 0]

C

Block name: C
Block type: Matrix Gain (Mask)

Matrix Gain.

Gain matrix:
[1 0 0 0 0 0; 0 0 1 0 0 0; 0 0 0 0 1 0]

TurnRateIn (w)

Block name: TurnRateIn (w)
Block type: Graph scope. (Mask)

Graph scope using MATLAB graph wi
Enter plotting ranges and line type.

Time range:
1800

y-min:
-.005

y-max:
.005

Line type (rgbw-.:*). Seperate each pl
'y-/g--/c-./w:/m*/ro/b+'

Ship 1 Track 1

Block name: Ship 1 Track 1
Block type: XY scope. (Mask)

XY scope using MATLAB graph window.
First input is used as time base.
Enter plotting ranges.

x-min:
-5.5

x-max:
0

y-min:
37.5

y-max:
42

Figure D.3. Simulink Model Parameters Two

ShipCInitSim.m

```
% ShipCInitSim.m
% 23 Oct 95
% This code defines the variables used by the simulink model.
% This is the initial placement of Vessel One
% [xposn (nm), xVel (nm/s), yposn (nm), yVel (nm/s), zposn (nm), zVel (nm/s)]

vesselVel =0.0028
crse=68

nautRad = crse *pi/180;%rad
[mathRad]=naut2mathRad(nautRad);

xVel =vesselVel*cos(mathRad )
yVel =vesselVel*sin(mathRad )

InitValue1=[-5.4600;xVel;38.89;yVel;0;0]% start at top of chart
                                        % y vel in degrees/sec
% 10 nm/hr  * 1hr/3600 sec = 0.0028  (nm or minutes)/sec
%
%10 knots * 1852 m/3600sec = 5.144 m/s

% Next we need to initialize the times for turns
%  Turn rates are calculated as follows
%
%           ||a||X g's * 9.8m/s^2
%        w= ------ =-------------------------- = [rads/sec]
%           ||v||speed (m/s)
%
%        for a tanker etc.  It should be about 3 min to turn 45 degrees
%        or 0.0044 rads/sec when doing about 10 knots

w = 0.0044           % use -ve w for CW turn and +ve w for CCW turn

startTurn1 = 122
endTurn1 = 330
initialValueTurn1 = 0
finalValueTurn1 = -w% CW turn

startTurn2 = 400
endTurn2 = 750
initialValueTurn2 = 0
finalValueTurn2 = w% CW turn

%  set the end time for simulation  >  30 min = 1800 sec

endtime              = 1900

rk45('SimVesselTrks',endtime)

%return
%  run this bit afterwards

save ShipCTrkData time lat lon
figure(1)
showChart1

%get(trk1Handle)
plot(lat,lon,'m.')
```

LIST OF REFERENCES

1. Ruthenberg, Thomas M., "Data Fusion Algorithm for the Vessel Traffic Services System: A Fuzzy Associative System Approach," Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1995.

2. Range Directorate Chesapeake Test Range, "System Design Document For The Coast Guard Vessel Traffic Service System," MIPR Number DTCG23-92-F-TAC111, Naval Air Warfare Center, MD, March, 1994.

3. Waltz, E. and Llinas, J., *Multi-Sensor Data Fusion*, Artech House Inc., Boston, MA, 1990.

4. Hall, David L., *Mathematical Techniques in Multisensor Data Fusion*, Artech House Inc., Norwood, MA, 1992.

5. Inter-National Research Institute (INRI), *Functional Description Document for Automated Dependent Surveillance (ADS)*, July 25, 1995.

6. Inter-National Research Institute (INRI), *Tdbm Service Application Programmer's Interface (API) For the Unified Build (UB) Software Development Environment (SDE)*, SPAWARSYSCOM SDE-API-TDBM-2.0.11.5, March 31, 1994.

7. Technical Information Exchange Meeting between Inter-National Research Institute, USCG and NPS, Reston, VA, November 3, 1995.

8. Telephonics, *MTE-2000 Marine Target Extractor Technical Manual*, Part No. 523854, Farmindale, NY, April 1, 1995.

9. Glenn, Ian N., "Multi-Sensor Multi-Target Tracking of Shipping Traffic," EC4980 Final Project, Naval Postgraduate School, Monterey, CA, May 25, 1995.

10. Bar-Shalom, Y. and Li, X., *Multitarget-Multisensor Tracking: Principles and Techniques*, 1995.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library, Code 013 Naval Postgraduate School Monterey, CA 93943-5101	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
4.	Prof. Murali Tummala, Code EC/Tu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	5
5.	Prof. Roberto Cristi, Code EC/Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
6.	Prof. Robert Hutchins, Code EC/Hu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
7.	Prof. Herschel H. Loomis, Jr.,, Code EC/Lm Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5121	1
8.	LCDR Michael Linzey P.O. Box 60 USCG EECEN Wildwood Crest, NJ 08260-0060	1
9.	LCDR Bobby Lam Coast Guard Navigation Center 7323 Telegraph Rd. Alexandria, VA 22315-3998	1

10. LT John Wood 2
P.O. Box 60
USCG EECEN
Wildwood Crest, NJ 08260-0060

11. Dr. Dwayne Presler 3
Inter-National Research Institute
12350 Jefferson Avenue, Suite 400
Newport News, VA 23602-6959

12. Brigadier General J.R. Marleau 1
Director General
Land Equipment Program Management
National Defence Headquarters
101 Colonel By Drive
Ottawa, Ontario
K1A 0K2

13. Dr. Laister 3
Director Soldier Systems Program Management
National Defence Headquarters
101 Colonel By Drive
Ottawa, Ontario
K1A 0K2

14. Mr. Dennis Guertin, DSIS 3 1
Director Scientific Information Services
National Defence Headquarters
101 Colonel By Drive
Ottawa, Ontario
K1A 0K2

15. Major Ian N. Glenn 2
Director Soldier Systems Program Management
National Defence Headquarters
101 Colonel By Drive
Ottawa, Ontario
K1A 0K2